ADVANCES IN TACTICAL & OPERATIONAL PLANNING FOR LESS-THAN-TRUCKLOAD CARRIERS

A Dissertation
Presented to
The Academic Faculty

By

Ahmad Baubaid

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial & Systems Engineering

Georgia Institute of Technology

December 2020

ADVANCES IN TACTICAL & OPERATIONAL PLANNING FOR LESS-THAN-TRUCKLOAD CARRIERS

Approved by:

Dr. Natashia L. Boland, Advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Martin W.P. Savelsbergh, Advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Alan L. Erera
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Chelsea C. White III
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Barrett W. Thomas
Tippie College of Business
The University of Iowa

Date Approved: November 19, 2020

The strongest of all warriors are these two — Time and Patience.

Leo Tolstoy

To my loving parents Faiza and Ali,

my late grandmother Salma,

my incredibly patient wife Fatmah,

and my little princess Alma.

ACKNOWLEDGEMENTS

This PhD experience has definitely been a challenging phase of my life both academically and personally. Between moving to a different country, starting a family, doing a PhD, and a pandemic towards the end, these past five years have been ones of tremendous growth, many mistakes and failures, along with, of course, the joys and successes that make it all worthwhile. I am full of appreciation for all of the people that I have encountered along the way, all of whom have helped me along and contributed to who I am, and consequently, this work. Although I will list a few names below, I am very mindful that there are many that I will surely, but not deliberately, forget to mention.

Firstly, I am very grateful to my advisors, Dr. Natashia Boland and Dr. Martin Savelsbergh, who were always supportive, available, patient, insightful, and understanding. I have learned a great deal from them not only about what it means to do research, but also what excellent teaching and advising looks like. Working with such prominent academics has been a great honor. I will also be forever thankful for the spontaneity of my first discussion with Dr. Boland about research opportunities during my time as a Master's student which has given me the opportunity of working with them – truly an example of how some of the best things that can happen to us are not necessarily ones we plan for. I would like to thank my thesis committee: Dr. Alan Erera, Dr. Chip White, and Dr. Barry Thomas for their insightful comments, discussion, and suggestions which have helped shape and guide this work. Furthermore, I would like to thank the many faculty members at ISyE that I have crossed paths with during my time here at Tech.

I am especially appreciative of my alma mater, King Fahd University of Petroleum and Minerals (KFUPM), for generously providing me with a scholarship to pursue both my Masters and my PhD. It is thanks to their generous support that I am fortunate enough to attend a world-renowned institution for my graduate studies. I am also grateful to the many professors at KFUPM whom I have closely interacted with and who played no small

part in helping me make the choice to pursue a PhD. In particular, I would like to thank Dr. Mohamed Ben-Daya, Dr. Malick Ndiaye, Dr. Mohammad Al-Habboubi, Dr. Salih Duffuaa, Dr. Shokri Selim, Dr. Mohammad Al-Durgam, and Dr. Fouad Al-Sunni. I would also like to thank my KFUPM friends and colleagues: Faisal Al-Ghamdi, Khaled Al-Shehri, Haitham Saleh, Yasser Al-Moghathawi, among many others.

I would also like to extend my thanks to the many friends I have made here at ISyE who have made this journey infinitely more enjoyable and have been there when things got a little tough. Special thanks to Will Lassiter, Amanda Chu, Reid Bishop, Reem Khir, Daniela Hurtado, and Adrian Rivera. With them, time always seemed to pass unbelievably quickly. I will forever cherish our time together working on homework assignments, our endless discussions about anything and everything, and the numerous moments of laughter and banter that we shared. I am sure that going through this experience together has forged a special bond that will last a lifetime.

Furthermore, I would like to express appreciation to the members in the Saudi Student Association at Georgia Tech. Special thanks to Yazeed Alaudah who has encouraged me to choose Georgia Tech for my graduate studies, constantly offered support and advice, and is a continuous source of inspiration. Furthermore, I would like to extend my thanks to Mohammad Al-Hassoun, Motaz Al-Farraj, Maad Al-Owaifeer, Mahmoud Al-Zahrani, Mohammad Nabhan, Abdullah Al-Amri, and Saeed Al-Abri.

Last but not least, I owe my deepest gratitude and most heartfelt thanks to my family. To my mother Faiza and my father Ali, thank you so much for your patience, love, sacrifice, and your constant support and encouragement. To my two sisters Alaa and Israa, and my brother Anas, thank you for your patience; I truly miss your company, and hope that with this thesis completed, there will be many more opportunities to spend time together and make up for these past few years. To my dear wife, Fatmah, I cannot thank you enough for your companionship, love, sacrifice, support, and your incredible patience despite living half the world away from your family. Without your help and unconditional support, I

cannot imagine completing this dissertation. To my daughter Alma (born halfway through this PhD journey), thank you for being not only a source of joy, but also one of distraction that constantly reminds me that there is more to life than work.

Lastly, as I am writing these words, the COVID-19 pandemic is still ongoing. This year has been quite the ride for everyone. Doing a PhD by itself isn't an easy undertaking, and trying to finish one amidst a pandemic while stuck at home with a toddler is a completely different experience (needless to say, this was not exactly what I signed up for). There are many people who have worked tirelessly over the course of this year to keep things ticking, and so I would be remiss not to express my gratitude to everyone who made this experience ever so slightly easier on us. Personally, the uncertainty involved with rapidly changing conditions while being away from our families and support networks – especially in the beginning of the pandemic – has been a major challenge, and the subsequent change in daily habits and routines has been hard for Fatmah, Alma, and myself to adapt to (definitely hardest on our little one). Overall, it has been ... interesting, to say the least. Hopefully, finally completing this thesis will put a beautiful end to an otherwise rough and challenging 2020, and that we will have more opportunities to have fun together in a much less anxiety-inducing 2021.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This thesis explores tactical and operational planning problems in the context of the Less-than-Truckload (LTL) industry. LTL carriers transport shipments that occupy a small fraction of trailer capacity, and, thus, rely on the consolidation of freight from multiple shippers to achieve economies of scale.

The first part of this thesis focuses on tactical planning operations of LTL carriers. In particular, in Chapter 2, we study the service network design problem confronted by LTL carriers ahead of an operating season. This problem includes determining: (1) the number of *services* (trailers) to operate between each pair of terminals, and (2) a *load plan* which specifies the sequence of transfer terminals that freight with a given origin and destination will visit. Traditionally, for every terminal and every ultimate destination, a load plan specifies a unique next terminal. We introduce the $\boldsymbol{p}$-alt model, which generalizes traditional load plans by allowing decision-makers to specify a desired number of next terminal options for terminal-destination pairs using a vector $\boldsymbol{p}$. We compare a number of exact and heuristic approaches for solving a two-stage stochastic variant of the $\boldsymbol{p}$-alt model. Using this model, we show that by explicitly considering demand uncertainty and by merely allowing up to *two* next terminal options for terminal-destination pairs in the load plans, carriers can generate substantial cost savings; cost savings that are comparable to those yielded by adopting load plans that allow for *any* next terminal to be a routing option for terminal-destination pairs. Moreover, by using these more flexible load plans, carriers can generate cost savings in the order of 10% over traditional load plan designs obtained by deterministic models.

The second part of the thesis shifts to an operational setting relating to how freight is routed through the carrier's service network. As the daily freight quantities handled by a carrier are uncertain, freight routes are dynamically adjusted on the day of operations. In Chapter 3, we introduce the Dynamic Freight Routing Problem (DFRP) which models the

problem of routing freight dynamically (in the presence of demand uncertainty) throughout the service network. We formally model this problem as a Markov Decision Process (MDP). To overcome the curses of dimensionality of the MDP model, we introduce an Approximate Dynamic Programming (ADP) solution approach for the DFRP. Subsequently, in Chapter 4, we describe a lookup table value function approximation (LT-VFA) mechanism for this ADP algorithm, and introduce and compare a number of aggregation approaches which use features of the post-decision states to aggregate the post-decision state space. Furthermore, since the decision subproblems encountered by the ADP algorithm are integer programs (IPs), we present a framework for integrating lookup tables into the decision subproblem IPs. This framework consists of: (1) a modeling approach for the integration of lookup table value function approximations into subproblem IPs to form extended subproblem IPs, (2) a solution approach, PDS-IP-Bounding, which decomposes the extended subproblem IPs into many smaller IPs and uses dynamic bounds to reduce the number of small IPs that have to be solved, and (3) an adaptation of the ϵ-greedy exploration-exploitation algorithm for the IP setting. Our computational experiments show that despite the post-decision state of the DFRP being high-dimensional, a two-dimensional aggregation of the post-decision space is able to produce policies that outperform standard myopic policies. Moreover, our experiments demonstrate that the PDS-IP-Bounding algorithm provides computational advantages over solving the extended subproblem IPs using a commercial solver.

Finally, in Chapter 5, we extend the work on DFRP by describing two parametric VFA variants for the DFRP ADP algorithm, namely, a linear value function approximation (L-VFA), and a neural network value function approximation (NN-VFA), both using features of the post-decision states to approximate the value function. We conduct computational experiments to compare the performance of all three VFA variants of the ADP solution approach on instances of the DFRP, and show that the L-VFA method outperforms its two counterparts in solution quality and the required computational effort.

CHAPTER 1
THESIS INTRODUCTION AND BACKGROUND

The trucking industry forms the backbone of the U.S. economy, as many other industries rely on their services for the timely transportation of shipments. It provides an important service to both businesses and consumers by transporting anything from raw materials to finished goods from business to business and from business to consumer.

Less-than-truckload (LTL) carriers are trucking companies that provide transportation for freight shipments that typically occupy less than 10% of trailer capacity. Consequently, these carriers rely on consolidation of freight from multiple shippers to make their operations economically viable. LTL carriers create consolidation opportunities by routing collected freight through a network of consolidation terminals as it is transported between origins and destinations. At each of these terminals, freight on incoming trailers is sorted and then consolidated into outgoing trailers that transport that freight to the next terminal on its journey to its final destination.

To ensure the timely delivery of freight and to improve service levels, these carriers must operate efficiently. This means that carriers must constantly adapt to changing regulations, economic conditions, and technological advancements in order to compete and continue providing reliable and low-cost services to their customers. Consequently, decisions at various planning levels (strategic, tactical, and operational) have to be made and constantly revised ([1]). This thesis focuses on some of the *tactical* and *operational* decisions that are made by LTL carriers.

1.1 LTL Carrier Operations

An LTL carrier's network is made up of two types of terminals: *end-of-line* (EOL) terminals, which are the terminals that serve as origins and/or destinations of freight, and

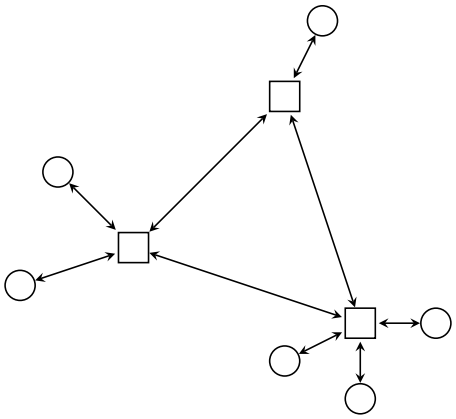


Figure 1.1: Example of an LTL carrier line-haul network; circles represent EOL terminals and squares represent BB terminals

breakbulk (BB) terminals, which in addition to being freight origins/destinations, are also consolidation points in the carrier's network. The set of EOL and BB terminals together comprise the carrier's *line-haul* network. An example sketch of a line-haul network can be seen in Figure 1.1.

During the day, each terminal dispatches trucks early in the day to deliver and pick up shipments from local customers that are served by that terminal. This daily local operation is known as the *city operation*. The collected shipments are brought to the terminal when the trucks return late in the day. Typically, there is not enough freight to economically justify dispatching a full trailer from the terminal directly to a destination terminal. For this reason, these shipments are loaded onto trailers that are dispatched to one or more nearby BB terminals to be consolidated with other shipments. After sorting the freight at the BB terminals, some of it is loaded onto trailers that are headed for other BBs, other freight is loaded onto trailers that are headed for nearby EOLs (containing shipments whose final destinations are consignees in the areas served by those EOLs), and some freight is loaded onto a city trailer for local delivery in a city operation tour. Therefore, a typical shipment might travel through two BB terminals on its way from its origin EOL to its destination EOL. The movement of freight *between* the terminals of the network is known as the *line-haul operation* of the carrier, and is the focus of this thesis.

1.2 LTL Carrier Decisions

To facilitate their line-haul operations, LTL carriers need to make many planning decisions. Depending on the length of the planning horizon considered, these decisions can either be strategic, tactical, or operational in nature. In this section, we provide a brief discussion and give examples for each category in the decision hierarchy. Although we give a brief description and some examples of strategic planning elements, we note that this thesis focuses only on tactical and operational elements.

1.2.1 Strategic Decisions

Strategic decisions related to the carrier's line-haul operations are long-term decisions that do not change over the span of several years. Examples of strategic decisions include determining the number of line-haul terminals to operate, their locations, and how they are connected (i.e. determining which terminals might send or receive freight to/from a given terminal).

1.2.2 Tactical Decisions

Tactical decisions related to the carrier's line-haul operations are medium-term decisions that may span several months (typically, an operating season). Decisions that need to made or revised for each operating season include determining the number of trailers needed for that operating season and their types/sizes, the trailer movement patterns and their schedules, and determining a *load plan* which dictates the possible paths that freight with a given origin and destination can take through the line-haul network. The load plan, therefore, dictates how freight will be consolidated in the line-haul network. We provide a more-in-depth discussion of load plans in Section 2.2.

In the literature, tactical decisions are captured and modeled through a class of problems known as *service network design* problems ([1, 2]). These problems are typically

formulated as Capacitated Multi-commodity Network Design (CMND) problems [3], and are known to be difficult to solve.

Although, in practice, predicted demand volumes are typically used to formulate these models, recently, attention has shifted to models that incorporate demand uncertainty. Including demand uncertainty, however, usually results in stochastic optimization models which are even more difficult to solve than their deterministic counterparts. Often, solutions to stochastic models yield insights into their structure which can be used to design heuristics for service network design problems ([4]).

1.2.3 Operational Decisions

Operational decisions related to the carrier's line-haul operations are short-term decisions, which are typically day-to-day decisions. These decisions are made in a highly dynamic environment typically by terminal managers. Examples include the various terminal operations (e.g. gate assignments for trailers, intra-facility activities, etc.), routing of freight shipments by deciding on which trailers to load them, and the closing of trailers docked at the terminal.

Although tactical planning problems in this field have received much attention, there is comparatively little research on operational problems. With recent technological advances, however, carriers are interested in streamlining and improving their operations to remain profitable and competitive. Furthermore, while carriers traditionally relied on simple rules for operating their terminals and on the knowledge and experience of their terminal managers, they are now looking towards embracing new technologies which enable them to make much more complex decisions that take into account the complete state of their operations at the time.

In addition to the complexity resulting from the scale of carriers' operations, demand uncertainty also poses a challenge. A decision made today about where to route freight to next can turn out to be a bad decision come tomorrow as newly arriving entering the system

may cause downstream congestions and lead to delays.

In Part II of this thesis, we focus specifically on freight routing decisions made in a dynamic and uncertain environment and contribute to the literature by designing a prototypical dynamic freight routing system, and addressing some of these issues.

1.3 Overview and Contributions

The remainder of this thesis consists of two parts. Part I is comprised of Chapter 2, and includes contributions to the tactical planning operations of LTL carriers, i.e. service network design. Part II is comprised of Chapters 3-5, and includes contributions to operational decision-making aspects, namely, dynamically routing freight shipments from origins to destinations in the presence of demand uncertainty.

In Chapter 2, we contribute to the literature on the service network design problem faced by LTL carriers. Traditionally, load plans only allow for a single next terminal option for freight at a particular terminal and with a particular destination (terminal-destination pair). However, in practice and because of demand uncertainty, freight can sometimes be diverted through a backup or alternate next terminal option if the capacity for the next terminal option becomes insufficient. Our contributions in this chapter include the following. Firstly, we present the $\boldsymbol{p}$-alt model which generalizes the traditional load plan design model by including a parameter that allows for controlling the number of next terminal options for terminal-destination pairs in the load plans. Secondly, we use this model to empirically show that, in an environment in which demand is uncertain, load plans in which a maximum of two next terminal options (given by stochastic $\mathbf{2}$-alt models), can yield most of the benefits of operating with a load plan in which all next terminal options are allowed for terminal-destination pairs. Thirdly, we contribute to the literature demonstrating the benefits of using stochastic ($\mathbf{1}$-alt and $\mathbf{2}$-alt) models that explicitly consider demand uncertainty in load plan design compared to the deterministic models adopted in practice. Lastly, we also compare the performance of a number of exact and heuristic approaches for solving

the SAA sample problems of the p-alt model in terms of runtime and solution quality. The content of this chapter has been published in [5].

In Chapters 3-5, we study the problem of dynamically routing freight under demand uncertainty in an LTL carrier's service network. Chapter 3 serves as an introductory chapter to the second part of the thesis. Contributions of this chapter are as follows. Firstly, in Chapter 3, we introduce the Dynamic Freight Routing Problem (DFRP) and present a formal MDP model for this problem. Secondly, we present an ADP solution approach for the DFRP which consists of an offline *training* phase in which the values of different states are learned via simulation to find a routing policy that is then used in an online *execution* phase.

Subsequently, in Chapter 4, we study the use of lookup table value function approximation (VFA) mechanisms in an IP setting. Our contributions in this chapter include the following. Firstly, we introduce a lookup table value function approximation (VFA) mechanism which is used as part of the ADP algorithm for the DFRP. Secondly, we propose a number of aggregation schemes that use features of the DFRP's post-decision states to aggregate lookup table entries. Thirdly, we contribute to the ADP literature by investigating and providing a framework for integrating lookup tables into integer programs (IPs) to solve the decision subproblem in the ADP algorithm. The framework consists of: (1) a modeling approach for the integration of lookup table value function approximations into decision subproblem IPs to form extended decision subproblem IPs, (2) a solution approach, PDS-IP-Bounding, which decomposes the extended subproblem IPs into many smaller IPs and uses dynamic bounds to reduce the number of small IPs that have to be solved, and (3) an adaptation of the ϵ-greedy exploration-exploitation algorithm for the IP setting. Lastly, we show that despite the high-dimensional post-decision states of the DFRP, using the proposed low-dimensional aggregation schemes can produce policies that outperform a standard myopic approach, and we also demonstrate the computational advantages of using the proposed PDS-IP-Bounding algorithm over solving the extended decision subproblem IPs

using a commercial solver.

In Chapter 5, we study the use of parametric VFAs for the DFRP's ADP algorithm. Our contributions in this chapter include the following. Firstly, we present a linear model and a neural net VFA mechanism (both use features or basis functions) for the ADP algorithm proposed in Chapter 4. Secondly, we compare all three variants of VFAs presented in this part of thesis in terms of solution quality and runtime. This contributes to the growing literature studying the use of neural networks as VFA mechanisms in ADP algorithms, and to the literature that compares parametric and non-parametric VFA approaches for ADP algorithms. Lastly, our computational experiments demonstrate that the linear model VFA outperforms the neural network and lookup table VFAs in both policy quality and solution time for the extended decision subproblem IPs encountered in the ADP algorithm.

Part I

Tactical Planning

CHAPTER 2

THE VALUE OF LIMITED FLEXIBILITY IN SERVICE NETWORK DESIGNS

2.1 Introduction

At the start of each operating season, LTL carriers must plan how that season's demand is to be served. This tactical planning process is referred to in the literature as *service network design* [1, 2]. In particular, for LTL carriers, service network design problems address the following aspects of the carriers' operations: (1) deciding the number of *services* (trailers) to operate in the network, their frequencies, and their types/sizes, and (2) deciding the *load plan* which dictates the sequence of terminals that freight with a given origin and destination should follow as part of its journey through the network. The load plan, therefore, is a key part of a carrier's tactical plan as it dictates how collected freight will be consolidated and routed on a day-to-day basis.

The *traditional* form of a load plan for an LTL carrier restricts freight movement such that freight arriving at a terminal and destined for some destination terminal d is always loaded onto trailers destined for a unique next terminal, regardless of the origin of that freight. Because load plans are devised with respect to *predicted* OD demands, some carriers recognize the benefit of having more flexible load plans in order to deal with the inherently uncertain demand. In particular, at some terminals, carriers might allow incoming freight destined for destination d to be loaded into trailers destined for *two* different next terminals. The choice of this additional routing option is usually determined after obtaining the traditional load plan, often based on past experience; these decisions are not integrated into the load plan design process. The additional routing option provides a backup option in case demand on the day is such that the routing option in the traditional load plan for that freight has insufficient capacity. (We present a formal discussion of these types of load

plans in Section 2.2.)

In this chapter, we demonstrate the benefit of explicitly designing more flexible load plans, and show that such plans are effective in dealing with demand uncertainty. In fact, we empirically show that these types of load plans with seemingly limited flexibility are as effective in dealing with demand uncertainty as load plans that allow freight with a given destination to be sent to *any* next terminal. As the latter setup is much more difficult to implement in practice, it is appealing for carriers to have a load plan with limited flexibility that is only marginally more complex to operate than a traditional one, but that still manages to yield the benefits of a much more flexible form of load plan.

To demonstrate these advantages, we formulate and solve a two-stage stochastic Capacitated Multi-commodity Network Design (CMND) problem [3]. In the first stage, scheduled capacity is installed in the network (i.e., the number of trailers to operate between each pair of terminals in the network) and an associated load plan is determined. The load plan respects the planner's desired level of flexibility (where flexibility is represented by the number of next terminals that a terminal can have for a particular destination). To accommodate realized daily demand, we allow the acquisition of additional capacity (i.e., outsourced trailers) in the second stage, if needed. Consequently, both stages of the model feature discrete decisions.

It is impractical to find optimal solutions to this two-stage stochastic CMND model, and, therefore, we employ a Sample Average Approximation (SAA) framework [6]. In this approach, we randomly sample a small number of demand scenarios, and a sample average is used to approximate the expected objective value in the original stochastic program. The resulting mixed-integer program, which we refer to as the *sample problem*, is also a CMND problem, and can be solved by deterministic optimization techniques. This process is repeated for a certain number of iterations, each time with a different sample, to obtain candidate first-stage solutions to the original problem. Each of these candidates is then evaluated on a much larger set of scenarios in order to obtain better estimates of

the recourse cost, i.e., the cost of acquiring outsourced trailers. Ultimately, the first-stage solution candidate with the lowest estimated expected total cost is selected as the final solution. The sample problem in each SAA iteration is a CMND problem, which is known to be difficult to solve, even if the sample problem contains just a single scenario [7, 8, 9]. Therefore, we evaluate and compare a number of solution approaches (both exact and heuristic) for solving the sample problem. After selecting a solution approach for the sample problems, we then conduct experiments to assess the benefits of limited flexibility load plans.

Our contributions in this chapter include the following:

- Introducing and studying a generalization of the traditional load plan design model by including a parameter that allows for controlling the level of flexibility desired in the load plan. We call the resulting model the $\boldsymbol{p}$-alt model.

- Showing that load plans with limited flexibility, i.e., two next destination options (given by stochastic $\mathbf{2}$-alt models), can yield most of the benefits of operating with a fully-flexible load plan when faced with uncertain demand (while adding only marginal operational complexity compared to traditional load plans).

- Expanding literature demonstrating the benefits of using models that explicitly consider demand uncertainty in load plan design compared to their deterministic counterparts (which are currently used in practice). While a similar comparison has been conducted for traditional load plans in [10], the model used was slightly different to our $\mathbf{1}$-alt model, as it takes elements such as facility handling capacity and time/service guarantees into account, whereas our model focuses on the load plan and the trailer capacity offered. In addition, we conduct this experiment for the more flexible load plans obtained using the new $\mathbf{2}$-alt model we introduce in this chapter.

- Comparing the performance of a number of exact and heuristic approaches for solving the sample problems of the $\boldsymbol{p}$-alt model in terms of runtime and solution quality

(within the context of a SAA approach).

The remainder of this chapter is structured as follows. Section 2.2 provides a brief introduction to the operations of LTL carriers and load plan designs. Section 2.3 presents an overview of relevant research. Section 2.4 formally defines the $\boldsymbol{p}$-alt model for the design of flexible load plans. Section 2.5 outlines the approaches we use to obtain high-quality solutions. Section 2.6 discusses the results of an extensive computational study, which explores algorithmic choices and analyzes the benefits of load plan variants. Finally, Section 2.7 presents our conclusions.

2.2 LTL Carrier Load Plans

As discussed in Section 1.1, we focus on the line-haul operations of LTL carriers in this thesis. In this chapter, our focus is particularly on service network design, and in this section, we provide an overview of load plans which are an important element in the service network design process for LTL carriers.

2.2.1 Traditional Load Plans

A load plan is a freight routing plan that specifies the sequence of terminals that each shipment will follow given its origin and destination. Traditionally, in their load plans, LTL carriers impose an *anti-arborescence* (or *in-tree*) structure for each destination d in the network [11]. In other words, all freight headed for destination d that is currently at some intermediate terminal i in the network is always directed to a *single* next terminal j, regardless of its origin. An example of this can be seen in Figure 2.1 where the solid lines represent a traditional load plan for destination d. Specifically, at terminal i, all freight headed for destination d is sent to terminal j. Note that this structure also implies that freight with a given OD pair will follow a unique path from origin to destination. Although this type of load plan is the most restrictive of the load plans we discuss, it is the simplest to operate locally at each terminal as workers need only check the final destination of an

incoming shipment to determine which outgoing trailer that shipment should be loaded onto.

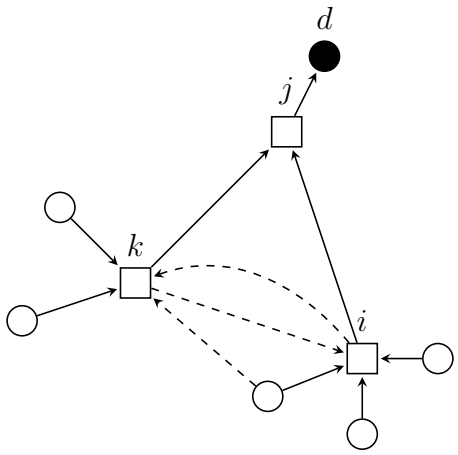


Figure 2.1: Example of a traditional load plan (solid lines) and a limited flexibility load plan (solid and dashed lines together) for a destination d in the line-haul network; Circles represent EOL terminals, squares represent BB terminals, and arrows indicate freight movement

2.2.2 Limited-Flexibility Load Plans

In what we refer to as *limited-flexibility* load plans, terminals can have up to two routing options for freight with a given destination. In practice, this would mean that there is a *primary* routing option, as well as an *alternative* option (or an "alt") which is used after exhausting the capacity offered on the primary on that day. However, in our work, we do not impose a priority for the primary option. We use the term limited-flexibility to contrast this with a *full-flexibility* load plan in which freight arriving at a given terminal can go to *any* next terminal.

If, on a given day, the scheduled trailer capacity available on the primary and the alt is not sufficient, then carriers can add additional outsourced trailers to either one at higher cost. Deals are typically negotiated with independent owner-operators in advance to help the carrier prepare for such eventualities. The dashed lines in Figure 2.1 represent an example of alts at some terminals for destination d (solid lines now represent the primaries).

In particular, all freight headed for destination d that is currently at terminal i can be sent to either terminal j (on the primary) or terminal k (on the alt) on a given day.

In this chapter, we show that limited-flexibility load plans can yield most of the benefits of full-flexibility load plans. As executing full-flexibility load plans involves complex workflows and requires a high-level of automation, limited-flexibility load plans provide an attractive alternative that provides similar benefits while being only marginally more complex to operate compared to traditional load plans.

2.2.3 Motivating Example

To motivate the benefits of introducing an alt at some terminals, consider the small example represented in the networks of Figure 2.2. In this example, the cost of operating a scheduled trailer moving between nodes i and k is $\frac{1}{2}$ (in either direction), while the cost of operating a scheduled trailer between nodes i and j as well as nodes j and k is 1 (also in either direction), and outsourced trailers cost 50% more than scheduled trailers. Furthermore, the underlying network structure is a complete graph. There are 3 commodities, and two possible realizations of demand, occurring with probabilities p_1 and p_2, described in Table 2.1.

Table 2.1: Demand values (in trailer-loads) for examples presented in Figures 2.2 and 2.3

Origin	Destination	Demand Realizations		Expected demand
		Realization 1 ($p_1 = 0.6$)	Realization 2 ($p_2 = 0.4$)	
i	j	2.00	1.00	1.60
k	i	1.30	0.30	0.90
k	j	1.00	2.70	1.68

Let us first design the service network using the expected demand values with the aim of minimizing the cost of the scheduled trailers. Note that we are using a deterministic model in this case even though we are operating in a stochastic environment. Thus, the existence of outsourced trailers is not explicitly considered – even though they may be required when

the uncertain demand realizes. Using the expected demand, we can obtain both a traditional load plan design (which we will denote as Determ-T) as well as a limited-flexibility design (which we will denote as Determ-L). The word 'design' here refers to decisions made about the scheduled trailers on an arc, and decisions about the load plan. For this particular instance, it turns out that the model chooses to offer the same scheduled services in both cases, namely two trailers on arc (i, j), one trailer on arc (k, j), and two trailers on arc (k, i). Thus, the total cost of both solutions is 4.5. The difference in these two designs, however, is in the underlying load plan. The load plan in the Determ-T solution stipulates that: freight originating at terminal i and destined for j can only be sent to j directly, freight originating at terminal k and destined for i can only be sent to i directly, and freight originating at terminal k and destined for j can only be sent to j directly. On the other hand the load plan in the Determ-L solution allows for freight originating at a terminal to transit through a second terminal on its way to its final destination.

In Figure 2.2, we show the solutions of both load plans under the two demand realizations. The solid, dashed, and dotted arcs are used to indicate the flows of commodities (i, j), (k, i), and (k, j), respectively. Numbers alongside each arc indicate the number of scheduled trailers on that arc. The plus sign is used on some arcs to indicate the need for a number of outsourced trailers. Numbers in between parentheses indicate total freight flows on the arcs. Thus, in the case of the Determ-T solution, we need an outsourced trailer on arc (k, i) to meet the demand in Realization 1 (at an additional cost of 0.75), and an additional trailer on arc (k, j) to meet the demand in Realization 2 (at an additional cost of 1.5). Therefore, the total expected cost of this solution is 5.55. In the case of the Determ-L design, we need an outsourced trailer on arc (k, i) again to meet the demand in Realization 1, but this time we do not need any outsourced trailers to meet the demand in Realization 2 since commodity (k, j) can now be split along the two paths from k to j. Thus, the total expected cost of this design is 4.95.

We now consider the use of a stochastic model to design the service network with the

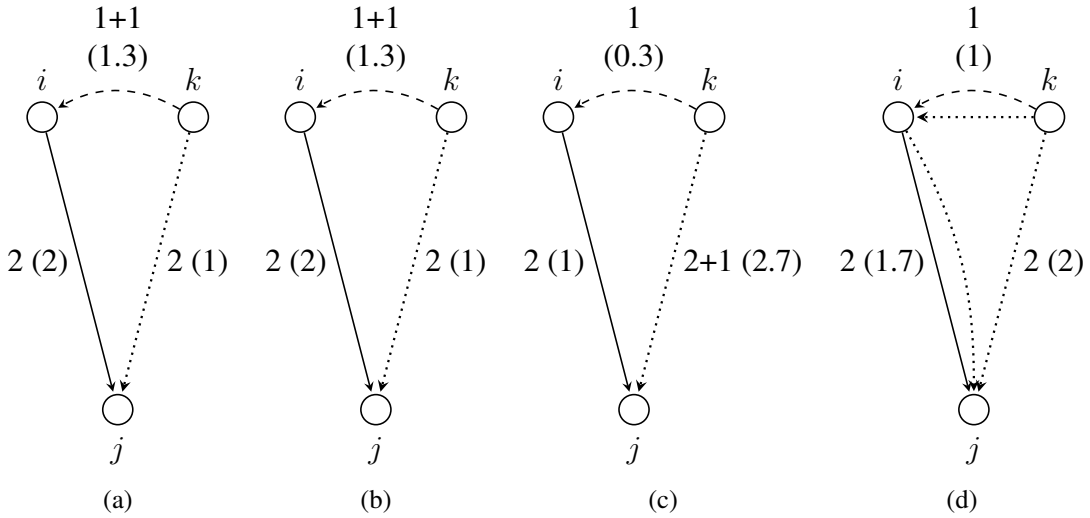


Figure 2.2: Deterministic solutions; (2.2a) shows the Determ-T solution in Realization 1; (2.2b) shows the Determ-L solution in Realization 1; (2.2c) shows the Determ-T solution in Realization 2; and (2.2d) shows the Determ-L solution in Realization 2

aim of minimizing the expected cost of operating trailers (*both* scheduled and outsourced) to service the demand. We only look at the case of allowing for a limited-flexibility load plan (which we denote as Stoch-L) and compare the performance of the Stoch-L solution with the Determ-L solution. For completeness, however, we show the total expected costs for all four cases including the case of a stochastic traditional load plan (i.e. Stoch-T) in Table 2.2.

Table 2.2: Total expected cost for cases considered in the motivating example

	Deterministic	Stochastic
Traditional	5.55	5.05
Limited-flexibility	4.95	4.55

Figure 2.3 shows the Stoch-L solutions under both realizations. We can immediately observe that the cost of the scheduled trailers in this solution is 3.5, which is less than the 4.5 of the Determ-L solution. In other words, the stochastic design does not over-commit to scheduled trailers because it recognizes that the use of outsourced trailers and smarter on-the-day consolidation will be cheaper, on average, in servicing the demand. In Realization

1, we need an outsourced trailer on arc (k, i) (at a cost of 0.75), and we need an outsourced trailer to service the demand in Realization 2 (on arc (k, j) at a cost of 1.5). Therefore, the total expected cost of the Stoch-L solution is 4.55 (lower than the 4.95 of its deterministic counterpart).

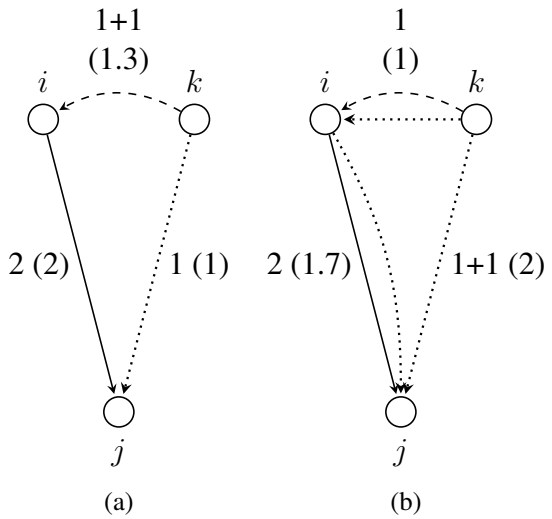


Figure 2.3: Stochastic solutions; (2.3a) shows the Stoch-L solution for Realization 1; (2.3b) shows the Stoch-L solution for Realization 2

Finally, although in practice the word "alt" usually just refers to the alternative next terminal option, for the remainder of this chapter, we will use the term "alt" to refer to *any* next terminal option, be it a primary or an alternative option, for the sake of brevity, and we will make no distinction between the two.

2.3 Literature Review

The $\boldsymbol{p}$-alt problem we present in Section 2.4 is a form of service network design (SND); a class of problems that has been well-studied in the literature and is applicable to multiple transportation settings. In particular, this SND model integrates LTL load plan requirements into the design process. For an overall review of SND, we refer the reader to [1] and [2]. Furthermore, [12] offers a brief review of the role of intermediate facilities in SND problems. In addition to trucking, this is a class of problems applicable to various trans-

portation settings including maritime transportation, e.g., [13, 14, 15], express shipment, e.g., [16, 17, 18, 19, 20, 21], rail, e.g., [22, 23, 24, 25], and multi-modal transportation systems, e.g., [26, 27, 28, 29, 30]. Generally, SND problems are formulated as Capacitated Multi-Commodity Network Design (CMND) problems [3] and are known to be intractable except for relatively small instances [7, 8, 9].

Many exact and heuristic approaches have been proposed in the literature for this class of problems. In terms of exact solution methods, a number of papers study cuts for these problems, e.g., [31, 32, 33, 34, 35, 36, 37, 38]. Some of the inequalities studied, including the *cut inequalities* we use in our solution approach (described in Section 2.5.2), are inequalities based on cutsets in the underlying graph, and heuristic procedures were proposed to separate for these cutset-based inequalities such as those found in [33, 34, 37, 38]. In addition to the exact methods, two common heuristic approaches frequently appearing in the literature are local search, e.g., [8, 7, 39, 40, 41], and slope scaling, e.g., [42, 43, 44], the latter of which we use in this work. Among the papers proposing heuristics, [8] presents a Tabu Search algorithm which searches a neighborhood using column generation and simplex-like pivoting moves in the space of the path flow variables. [7] defines a neighborhood consisting of cycles which can be used to reroute the flows of multiple commodities, and then embeds this neighborhood in a Tabu Search algorithm. This was further combined with a path-relinking algorithm in [39]. On the other hand, slope scaling is a heuristic approach introduced in [42] that iteratively solves linear approximations of the original problem formulation, adjusting the costs in each iteration, in an attempt to arrive at good feasible solutions. [43] introduces a slope scaling algorithm which integrates a Lagrangean perturbation scheme with some metaheuristic elements.

A number of recent papers on SND study how demand uncertainty affects network designs, and develop heuristic approaches for these problems. Specifically, [4] investigate the effects of demand uncertainty on network designs and the structural differences of these designs compared to their deterministic counterparts. They observe that consolida-

tion is a natural byproduct that arises in designs to hedge against the uncertainty of demand. [45] and [46] develop metaheuristic solution approaches for the stochastic service network design problem. In search of network flexibility, [9] propose a stochastic network design model that allows rerouting or rescheduling of vehicles in the second stage of the stochastic program. Finally, [47], [48], and [49] study the quality of deterministic solutions and their "upgradeability" to solutions of the stochastic scheduled service network design problem.

Relating to SND for LTL carriers, load plan design for these carriers was first studied in [50] and later that was followed by [51], [52], and [11]. In these papers, the problem was defined and formulated as a mixed-integer program, and local improvement heuristics were suggested to solve large-scale instances for a large U.S. LTL carrier. These papers were the first to explicitly consider the in-tree load plan structures in the network designs. To model service requirements more accurately, [44] present a service network design formulation defined on a time-expanded network, and a heuristic which combines slope scaling and column-generation (where the columns are the in-tree load plans) to arrive at high-quality solutions for large-scale real-life instances for a large carrier. More recently, [40] investigate the cost savings generated by varying the load plan by day of week to increase the flexibility of designs, and note that such flexibility generates approximately 6.5% in savings to the carrier. Moreover, [40] and [41] develop methods to improve load plan designs through IP-based local search techniques which they use to generate significant cost savings for a U.S. carrier.

There is also a number of papers dealing with load planning for time-definite freight delivery common carriers. Time-definite carriers provide guaranteed door-to-door pickup and delivery services for small shippers, typically publishing rates, routes, and schedules for the general public. They consolidate shipments and utilize load plans with an in-tree structure similar to that of LTL carriers. [53] presents the (deterministic) freight routing problem for these carriers along with two approaches to solve this problem, a Lagrangian relaxation approach and an implicit enumeration algorithm with ϵ-optimality (IE-ϵ). [10]

presents a two-stage stochastic load planning problem for time-definite freight common carriers facing uncertain demand, which was later expanded into a multi-stage model in [54]. Their models (in particular, the in-tree load planning requirement and the use of additional trailers as a recourse) are similar to the traditional load plan variant of the model we present in this chapter. However, their models contained additional constraints relating to service commitment, facility handling capacity, and trailer balancing. Furthermore, demand was assumed to be given by a finite discrete distribution, whereas we work with continuous distributions. Both papers develop heuristics to solve their respective models, and use them to compare costs of deterministic and stochastic solutions for small instances selected as subsets of the network of a large Taiwanese carrier. They conclude that using a stochastic model yields solutions with a lower expected operating cost compared to solutions of deterministic models.

2.4 Problem Description

We formulate the stochastic $\boldsymbol{p}$-alt problem as a two-stage stochastic CMND problem with integer variables for service selection and an additional $\boldsymbol{p}$-alt constraint. We use the boldface parameter $\boldsymbol{p}$ to represent a vector specifying the desired number of alts allowed for each terminal-destination pair. This is a general form for this parameter as, in practice, the number of alts desired for a terminal-destination pair may depend, for example, on the size of the terminal and the volume moving through that terminal to the destination.

In the first stage, the aim is to design a service network with the desired $\boldsymbol{p}$-alt structure. A *design* specifies the number of scheduled trailers on each arc in the network, i.e., the installed capacity, as well as a load plan with the desired alt structure, i.e., the potential sequences of terminals that freight with a given origin and destination can follow. In the second stage, this design is used to satisfy realized demand, possibly introducing outsourced trailers obtained at a higher cost. Hence, the design variables (first stage variables) are independent of demand realizations, whereas the variables relating to commodity flows

and outsourced capacity are realization-dependent.

We define the problem on a carrier's line-haul network which is comprised of EOL and BB terminals. Only EOL terminals serve as origins or destinations in this model. This is without loss of generality, as "EOL" copies of BB terminals can be added to the network if freight also originates from or can be destined to certain BB terminals. Commodities in this problem represent shipments that have the same OD pair, and associated with each is a certain quantity representing its demand. The problem is to minimize the total expected cost which consists of the cost of scheduled trailers, plus the expected cost of outsourced trailers. We assume here a homogeneous fleet of vehicles in both stages of the problem, although the model can be easily extended to more general fleets.

Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be a digraph that represents the terminal network of the carrier with $\mathcal{N} = \mathcal{B} \cup \mathcal{E}$, where $\mathcal{B}$ and $\mathcal{E}$ represent the sets of BB and EOL terminals, respectively. Define the set of commodities $\mathcal{K}$, as a subset of the set of all possible EOL pairs for which there might be demand, i.e., $\mathcal{K} \subseteq \{(o, d) : o, d \in \mathcal{E}, o \neq d\}$. Let $o_k, d_k \in \mathcal{E}$ denote the origin and ultimate destination for commodity k, respectively. For notational convenience, define $\mathcal{D}$ as the set of all EOL terminals that are ultimate destinations for at least one commodity, i.e., $\mathcal{D} = \{d_k : k \in \mathcal{K}\}$, and $\mathcal{K}(d)$ as the set of all commodities with ultimate destination d.

Let p_{id} be the number of alts allowed in the load plan design for terminal i and destination d. We define c_{ij} as the cost of operating a scheduled trailer on arc (i, j), and $\hat{c}_{ij} > c_{ij}$ as the cost of operating an outsourced trailer operated on that same arc. Define Q to be the uniform trailer capacity (for both scheduled and outsourced trailers). Let $\Omega \subseteq \mathbb{R}_+^{|\mathcal{K}|}$ represent the set of random commodity demands that are possible for commodities in $\mathcal{K}$ (this set can be either discrete or continuous), with $\omega \in \Omega$ representing a particular realization. Finally, define q_k^ω as the quantity of commodity k in realization ω.

Our decision variables are as follows:

r_{ij} = number of *scheduled* trailers to operate on arc $(i,j) \in \mathcal{A}$,

z_{ij}^{ω} = number of *outsourced* trailers to operate on arc $(i,j) \in \mathcal{A}$ in realization ω,

x_{ijk}^{ω} = flow on arc $(i,j) \in \mathcal{A}$ for commodity $k \in \mathcal{K}$ in realization ω,

$$y_{ijd} = \begin{cases} 1, & \text{if commodities with destination } d \text{ can use arc } (i,j) \in \mathcal{A} \text{ as an alt,} \\ 0, & \text{otherwise.} \end{cases}$$

2.4.1 Mathematical Formulation

The stochastic ***p*-alt model** is:

$$\min \sum_{(i,j)\in\mathcal{A}} c_{ij} r_{ij} + \mathbb{E}_{\omega}\left[\tilde{Q}(r,y,\omega)\right], \tag{2.1a}$$

$$\text{s.t.} \sum_{(i,j)\in\delta^{+}(i)} y_{ijd} = \min\{p_{id}, \left|\delta^{+}(i)\right|\}, \qquad \forall d \in \mathcal{D},\ i \in \mathcal{N}, i \neq d, \tag{2.1b}$$

$$y_{ijd} \in \{0,1\}, \qquad \forall d \in \mathcal{D},\ (i,j) \in \mathcal{A},, \tag{2.1c}$$

$$r_{ij} \in \mathbb{Z}_{+}, \qquad \forall (i,j) \in \mathcal{A}, \tag{2.1d}$$

where

$$\tilde{Q}(r,y,\omega) = \min\{ \sum_{(i,j)\in\mathcal{A}} \hat{c}_{ij} z_{ij}^{\omega} \ :\ (z^{\omega}, x^{\omega}) \in \mathcal{P}(r,y,\omega)\}, \tag{2.1e}$$

and

$$\mathcal{P}(r,y,\omega) = \{(z^\omega, x^\omega) \ : \ \sum_{k\in\mathcal{K}} x^\omega_{ijk} \le Q(r_{ij} + z^\omega_{ij}), \qquad \forall (i,j) \in \mathcal{A}, \quad (2.1f)$$

$$\sum_{(i,j)\in\delta^+(i)} x^\omega_{ijk} - \sum_{(j,i)\in\delta^-(i)} x^\omega_{jik} = \begin{cases} q^\omega_k, & \text{if } i = o_k, \\ -q^\omega_k, & \text{if } i = d_k, \\ 0, & \text{otherwise,} \end{cases} \qquad \forall i \in \mathcal{N},\ k \in \mathcal{K}, \quad (2.1g)$$

$$x^\omega_{ijk} \le q^\omega_k y_{ijd}, \qquad \forall d \in \mathcal{D},\ (i,j) \in \mathcal{A},\ k \in \mathcal{K}(d), \quad (2.1h)$$

$$x^\omega_{ijk} \ge 0, \qquad \forall (i,j) \in \mathcal{A},\ k \in \mathcal{K}\ , \quad (2.1i)$$

$$z^\omega_{ij} \in \mathbb{Z}_+, \qquad \forall (i,j) \in \mathcal{A}\}. \quad (2.1j)$$

The first stage determines the design of the network, which includes determining the number of scheduled trailers to operate on each arc and choosing up to p_{id} alts for each terminal-destination pair. The objective function (2.1a) minimizes the total expected cost which consists of the cost for installing scheduled capacity as well as the cost of acquiring additional capacity. Constraints (2.1b) are the $\boldsymbol{p}$-alt constraints that determine the number of alts allowed. Written in this form, the constraint ensures that each terminal-destination pair is assigned p_{id} alts for each commodity wherever possible, and as many alts as possible otherwise. Although there is no reason to select a value for p_{id} that is greater than the outdegree of a node i, writing the constraint in this form allows us to conveniently refer to the traditional load plan model as a $\mathbf{1}$-alt model, and a limited-flexibility load plan as a $\mathbf{2}$-alt model, where $\mathbf{1}$ and $\mathbf{2}$ are the vectors of all ones and all twos, respectively.

In the second stage, we deal with realized demand. Given our first stage design decisions, we attempt to satisfy all demand using the scheduled capacity or, if necessary, with additional outsourced capacity. The objective function (2.1e) minimizes the cost of installing additional capacity measured in the cost of adding additional trailers. Con-

straints (2.1f) ensure that sufficient arc capacities are available to accommodate freight flows. Constraints (2.1g) are the standard flow balance constraints, and ensure that all demand is met. Constraints (2.1h) ensure that flows are compatible with the chosen load plan design by only allowing a commodity k to flow on an arc (i, j) if that arc is chosen as an alt for terminal i and for destination d_k. Note that we use the disaggregated form of these constraints, as they provide a tighter formulation and performed better in our computational experiments. Although this means many more constraints in our model, for the sizes of instances we consider, this was not a major concern.

It is worth noting that an important feature of our setup is that we only allow outsourced trailers to be operated on arcs that are alts for at least one destination. From a practical perspective, this restriction helps carriers plan in advance for the services that they expect they would need when demand realizes enabling them to negotiate better deals for obtaining extra capacity from independent owner-operators. Although this restriction is not explicitly enforced in the model as a constraint, the combination of the objective function (2.1e), constraints (2.1f) and (2.1h) achieves the desired outcome.

Although most models for finding a standard $\mathbf{1}$-alt load plan presented in the literature are path-based models, using an arc-based model is far more convenient when seeking to find a $\mathbf{2}$-alt load plan. Note that whereas a $\mathbf{1}$-alt structure implies a single path for an OD pair, a $\mathbf{2}$-alt structure allows many paths for an OD pair (with the exact number depending on the configuration of the alts).

To benchmark the cost savings of $\mathbf{2}$-alt designs, we will make use of a similar stochastic model that does not impose any structure on the freight flows. We will refer to this model as the stochastic **Infinite-Alt** model. Note that for a given network, setting all the values of the vector $\boldsymbol{p}$ to $\max_{i \in \mathcal{N}}\{|\delta^+(i)|\}$ makes the alt constraint redundant. Therefore, the stochastic infinite-alt model represents a relaxation of the stochastic $\boldsymbol{p}$-alt model and serves as a natural benchmark for performance. For the sake of completeness, the stochastic

Infinite-Alt (IA) model is as follows:

$$\min \sum_{(i,j)\in\mathcal{A}} c_{ij} r_{ij} + \mathbb{E}_{\omega}\left[\tilde{Q}(r,\omega)\right], \tag{2.2a}$$

$$\text{s.t. (2.1d)}, \tag{2.2b}$$

where

$$\tilde{Q}(r,\omega) = \min \sum_{(i,j)\in\mathcal{A}} \hat{c}_{ij} z^{\omega}_{ij}, \tag{2.2c}$$

$$\text{s.t. (2.1f), (2.1g), (2.1i), and (2.1j).} \tag{2.2d}$$

Note the omission of the y variables in the above model.

2.5 Model Solution

Given that a two-stage stochastic CMND problem (with integer variables in both stages) is very difficult to solve, we employ Sample Average Approximation (SAA) [6] as the overarching solution framework. A SAA method consists of a number of iterations, each requiring the solution of a deterministic *sample problem*. In our setting, the sample problem is a deterministic CMND problem. Because the sample problem is difficult to solve, we investigate the use of both exact and heuristic approaches.

The rest of this section is structured as follows. First, we describe the SAA framework that we employ. Then, we present the *cut inequalities* for the CMND problem and discuss the methods we use to separate for these inequalities. Finally, we outline the different exact and heuristic approaches we use to solve the sample problem in each iteration of the SAA framework.

2.5.1 Sample Average Approximation (SAA)

Given that the set of possible scenarios Ω is usually very large (and possibly infinite), SAA uses sampling to sidestep this issue. In SAA, the set Ω is replaced by $\mathcal{S}$, a small set of randomly sampled scenarios of size $N \ll |\Omega|$, and the expectation $\mathbb{E}_\omega \left[\tilde{Q}(r, y, \omega)\right]$ is approximated by the sample average obtained using these N scenarios, $\frac{1}{N}\sum_{\omega\in\mathcal{S}} \tilde{Q}(r, y, \omega)$. This results in the following deterministic model which we will refer to as the *sample problem*:

$$\min \sum_{(i,j)\in\mathcal{A}} c_{ij} r_{ij} + \frac{1}{N}\sum_{\omega\in\mathcal{S}}\sum_{(i,j)\in\mathcal{A}} \hat{c}_{ij} z^\omega_{ij}, \tag{2.3a}$$

$$\text{s.t.} \quad (2.1b) - (2.1d),\ (z^\omega, x^\omega) \in \mathcal{P}(r, y, \omega),\ \forall \omega \in \mathcal{S}. \tag{2.3b}$$

SAA repeats this process M times, using M different N-samples, and by solving the sample problem in each one, it obtains a set of M candidate first-stage decisions (or *designs*), $(\hat{r}^m, \hat{y}^m), m = 1, \ldots, M$. To get a better estimate of the recourse cost, each of these designs is evaluated on $N' \gg N$ sampled scenarios. The total cost of a design, $(\hat{r}^m, \hat{y}^m)$, can then be approximated by

$$\sum_{(i,j)\in\mathcal{A}} c_{ij}\hat{r}^m_{ij} + \mathbb{E}_\omega \left[\tilde{Q}(\hat{r}^m, \hat{y}^m, \omega)\right] \approx \sum_{(i,j)\in\mathcal{A}} c_{ij}\hat{r}^m_{ij} + \frac{1}{N'}\sum_{n=1}^{N'} \tilde{Q}(\hat{r}^m, \hat{y}^m, \omega^n). \tag{2.4}$$

where $\omega^n, n = 1, \ldots, N'$, is the set of sampled scenarios for evaluation.

For completeness, we include here a description of a *generic* SAA procedure for a minimization problem [6]:

1. Select M, the number of iterations for the procedure. Select N, the sample size for the sample problems. Select N', the sample size for the evaluation subproblems.

2. For $m = 1, \ldots, M$,

   2.1 Generate N scenarios of Ω, and solve the resulting sample problem. Let $\hat{x}^m$

be the first-stage optimal solution of the sample problem (not to be confused with the flow variables particular to the $\boldsymbol{p}$-alt problem), and v_N^m be the sample problem's objective value.

3. Calculate

$$\bar{v}_N = \frac{1}{M}\sum_{m=1}^{M} v_N^m,$$

a statistical estimate for a lower bound on the optimal value of the true problem, and its variance

$$\sigma^2_{\bar{v}_N} = \frac{1}{(M-1)M}\sum_{m=1}^{M}(v_N^m - \bar{v}_N)^2.$$

4. For any feasible first-stage solution $\hat{x}$, calculate a statistical estimator for an upper bound on the optimal value of the true problem, by generating N' scenarios from Ω (independently from the N used in the sample problem), then computing

$$\hat{v}_{N'}(\hat{x}) = \frac{1}{N'}\sum_{n=1}^{N'} G(\hat{x}, \omega^n)$$

where $G(\hat{x}, \omega^n)$ is the total objective value (first-stage and recourse) obtained by solving the evaluation subproblem for scenario ω^n and feasible solution $\hat{x}$. The variance of this estimator can be computed as

$$\sigma^2_{\hat{v}_{N'}(\hat{x})} = \frac{1}{(N'-1)N'}\sum_{n=1}^{N'}(G(\hat{x}, \omega^n) - \hat{v}_{N'}(\hat{x}))^2.$$

5. Select $\hat{x}^* \in \underset{m\in\{1,\ldots,M\}}{\arg\min}\{\hat{v}_{N'}(\hat{x}^m)\}$.

6. Calculate the estimate for the optimality gap

$$\hat{v}_{N'}(\hat{x}^*) - \bar{v}_N, \tag{2.5}$$

and its variance,

$$\sigma^2_{\hat{v}_{N'}(\hat{x}^*)} + \sigma^2_{\bar{v}_N}. \tag{2.6}$$

In some variants of SAA, an additional step is added to check if the optimality gap estimate given by Equation (2.5) is sufficiently small. If not, the entire procedure can be repeated, with one of M, N, or N' increased (typically, N). For the stochastic $\boldsymbol{p}$-alt model we presented, the pseudocode for the SAA framework we use is outlined in Appendix A.1.1.

2.5.2 Cut Inequalities

The *cut inequalities* [33] we present here will be used in some of the solution methods for solving the sample problem (discussed in the next subsection). We describe these inequalities in the context of the sample problem of the $\boldsymbol{p}$-alt model given by (2.3).

These inequalities are generic cuts that can be used for many network flow models. They stipulate that for any given cut in the graph, the capacity crossing the cut should be enough to serve the demand crossing that cut. To adapt these inequalities to our problem setting, let V be any cutset in $\mathcal{G}$ and define $d^\omega(V, \bar{V})$ to be the total demand that has to traverse the cut defined by V in scenario ω, i.e., the demand of commodities whose origins and destinations are on different sides of the cut. Then, we write the cut inequalities for scenario ω as

$$\sum_{(i,j)\in\delta^+(V)} (r_{ij} + z^\omega_{ij}) \geq \frac{d^\omega(V, \bar{V})}{Q}, \qquad \forall\, V \subset \mathcal{N}. \tag{2.7}$$

Using the integrality of the trailer variables in both stages of the problem, we can obtain a stronger version of this inequality by simply rounding up the right hand side of (2.7). This yields the set of inequalities:

$$\sum_{(i,j)\in\delta^+(V)} (r_{ij} + z^\omega_{ij}) \geq \left\lceil \frac{d^\omega(V, \bar{V})}{Q} \right\rceil, \qquad \forall\, V \subset \mathcal{N},\ \omega \in \mathcal{S}. \tag{2.8}$$

These inequalities are usually violated by LP relaxation solutions to multi-commodity net-

work flow problems [33], and therefore, significantly strengthen the formulation. Note, however, that there is an exponential number of such inequalities, namely $\mathcal{O}(N \cdot 2^{|\mathcal{N}|})$. Therefore, enumerating all such inequalities is impractical. Furthermore, the separation problem for these inequalities is known to be NP-Hard [33], and separation heuristics are typically used ([33, 34, 37]).

2.5.3 Solving the Sample Problem

In each iteration of the SAA method, a sample problem given by mixed-integer linear program (2.3) has to be solved. Although simpler than the original stochastic problem, it is still difficult to solve to optimality in a reasonable amount of time.

In this section, we discuss the approaches we have investigated for solving the sample problem. We have considered *exact approaches*, in which we retain the integrality of both stages of the sample problem, as well as *heuristic approaches*, in which we relax the integrality of the second-stage variables. Note that by retaining the integrality of the first-stage variables, we always obtain a feasible design. By relaxing the second-stage variables, we hope to make the problem easier, but still obtain good-quality designs. As we found the evaluation subproblems in the SAA algorithm to be relatively easy to solve, we solve them exactly in both approaches.

In all except one of the approaches we have investigated, we use cut inequalities to strengthen the formulation (exact approaches) and/or to mitigate the effect of relaxing the second-stage variables (heuristic approaches). The inequalities are added either *statically*, i.e., they are added up front, or *dynamically*, i.e., they are added only when violated (during the solution process).

In the static SELECT approach, we add only a select few cut inequalities to the model. By exploiting the structure of the instances, we generate a small number of cuts that, hopefully, correspond to important cutsets. By generating the cuts upfront and by keeping the number of cuts small, most of the solution time can be spent exploring the search tree.

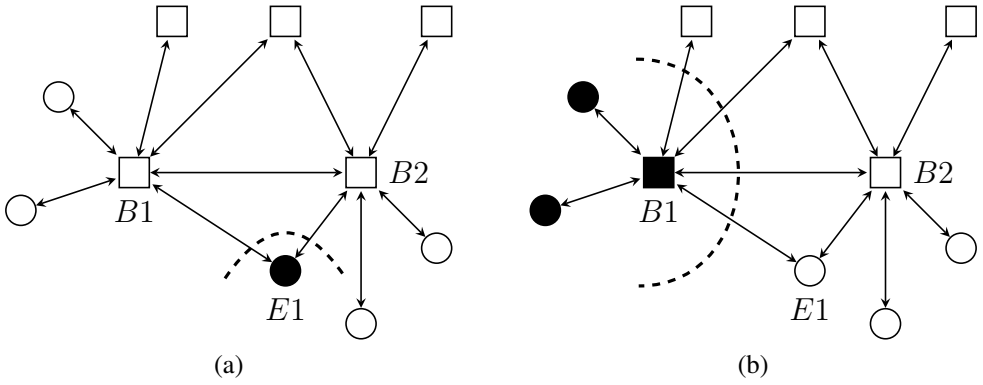
In our instances, an EOL terminal is connected to either one or two BB terminals, where an EOL terminal connected to two BB terminals is referred to as a *multi-loading* EOL (a multi-loading EOL will have a primary BB terminal, taken to be the nearest BB terminal, and a secondary BB terminal).

To illustrate the types of cuts we add in this approach, consider the example in Figure 2.4. In this example, all the EOL terminals that connect to terminals B1 and B2 are shown. Note, in particular, that EOL terminal E1 is a multi-loading EOL since it is connected to two BB terminals, with B2 being its primary BB.

The first types of cuts we consider are the cut inequalities corresponding to EOL cutsets. However, for an EOL terminal that connects to only one BB terminal, the cutset contains only a single arc, and the flow on the arc will be the total demand originating from (or destined for) the EOL terminal. The solver should be able to deduce this immediately from the flow balance constraints (2.1g), and then use a combination of the capacity constraints (2.1f) and the integrality of the trailer variables to arrive at the cut inequality corresponding to this cutset. Therefore, for EOLs, we will only add the cut inequalities corresponding to multi-loading EOLs (in both directions). An example can be seen in Figure 2.4a.

The next type of cuts we consider are the cut inequalities corresponding to cutsets containing a BB terminal along with its EOLs. There are two options here: (1) adding all EOLs connected to that BB in the cutset, (2) adding only EOLs for which that BB terminal is a primary BB. Preliminary experiments on small instances showed that the latter outperformed the former in terms of average solution time for the sample problems. Therefore, we add cut inequalities corresponding to BB terminals along with connected EOLs that have that BB as their primary BB to the model (we add the cut inequality in both directions here as well). An example of this can be seen in Figure 2.4b.

In the static ENUM approach, all cutsets of size less than or equal to a given maximum size, M^{Enum}, are generated and the cut inequalities corresponding to those cutsets as well



(a) (b)

Figure 2.4: Illustration of selected cuts in the Exact-Select method; squares represent BB terminals and circles represent EOL terminals, and dashed lines indicate the cuts in each figure while the shaded nodes indicate nodes that are in the cutset; (2.4a) shows the cut corresponding to the multi-loading EOL E1; (2.4b) shows the cut corresponding to the BB terminal B1 and the EOL terminals for which it is a primary BB (note the exclusion of E1).

as to their complements are added to the model. We are careful to only add inequalities corresponding to *relevant* cuts, where relevant cuts are defined as those that separate at least one commodity's origin from its destination.

In the dynamic RANDCUT approach, the separation heuristic randomly generates a prespecified number of cutsets, $RCMaxIter$, for each scenario and checks if they are violated. Specifically, the heuristic randomly selects nodes in the graph to be in the cutset with each node having probability $\frac{1}{2}$ of being selected. If the cut inequality corresponding to the resulting cutset or its complement is violated, then it is recorded along with its violation. When $RCMaxIter$ cutsets have been generated, the recorded cutsets for the scenario are sorted in non-increasing order of their violations and the γ^{RC} most violated cut inequalities are added to the model. The pseudocode for RANDCUT is given in Appendix A.1.2.

In the dynamic RGCONTRACTION approach, a randomized greedy contraction heuristic is used to find violated cut inequalities. The heuristic combines and builds on ideas found in [33, 34, 37] and [38]. The heuristic is based on the observation that cut inequalities have a higher chance of being violated when the arcs in the cut have smaller slacks with respect to constraint (2.1f) and when they have smaller fractional parts with respect to

the right hand side of that constraint, i.e., $(\underline{r}_{ij} + \underline{z}^{\omega}_{ij}) - \lfloor (\underline{r}_{ij} + \underline{z}^{\omega}_{ij}) \rfloor$.

The heuristic iteratively contract arcs of the graph whose slacks and/or fractional parts are large. What remains at the end of this contraction process should be a graph whose arcs have small slacks and/or fractional values, and cuts in that graph are more likely to have violating cut inequalities. More specifically, arcs in the graph are sorted in non-increasing order of their slack values, and then by non-increasing order of their fractional part values. A GRASP-inspired selection procedure is used to identify the next arc in the sorted list to contract. Namely, we form a Restricted Candidate List (RCL) of a predetermined size, l, containing the first l arcs in the sorted list, and we randomly select an arc to contract from that RCL. Our experiments have shown that there is benefit to be gained in using this selection procedure as opposed to always contracting the first arc in the list.

The contraction is terminated whenever the shrunk graph has a certain number of remaining nodes, $\mathcal{N}^{final}$. All cuts on the shrunk graph are enumerated and checked for violations. Cutsets whose inequalities are violated are recorded (along with their violation), and this process is repeated for a pre-specified number of iterations, $RGMaxIter$, to increase the chance of finding violated cut inequalities. Finally, all collected cutsets are sorted in non-increasing order of their violations, and a pre-specified number of these, γ^{RG}, is added to the model. The pseudocode for RGCONTRACTION is given in Appendix A.1.3.

Note that when contracting an arc, (u, v), if for some node $w \neq u, v$ both the arcs (u, w) and (v, w) (or (w, u) and (w, v)) are in $\mathcal{G}$, then the slack of the new arc, (u, w) (or (u, w)) say, is the sum of the slacks on the arcs (u, w) and (v, w) (or (w, u) and (v, w)), and the fractional part of the new arc is the fractional part of the sum of the two fractional parts.

A final heuristic approach provides a different way of mitigating the effects of the relaxation of the second-stage variables, namely using dynamic *slope scaling* ([42]). Since relaxing the second-stage variables under-approximates the cost in the objective function, here we compensate for that by introducing cost multipliers that will be iteratively adjusted using dynamic slope scaling. Let $\rho^{t}_{ij\omega}$ be the cost multiplier for arc (i, j) in iteration t and

scenario ω of the slope scaling algorithm, and solve

$$\min \sum_{(i,j)\in\mathcal{A}} c_{ij} r_{ij} + \frac{1}{N} \sum_{\omega\in\mathcal{S}} \sum_{(i,j)\in\mathcal{A}} \rho^t_{ij\omega} z^\omega_{ij} \tag{2.9}$$

$$\text{s.t. (2.1b)} - \text{(2.1d)}, \ (z^\omega, x^\omega) \in \mathcal{P}_{LP}(r, y, \omega), \ \forall \omega \in \mathcal{S}$$

where $\mathcal{P}_{LP}(r, y, \omega)$ denotes the LP relaxation of $\mathcal{P}(r, y, \omega)$.

In each iteration of the slope scaling algorithm, if the stopping criteria are not met, we adjust the cost multipliers of each arc and each scenario as follows:

$$\rho^{t+1}_{ij\omega} = \begin{cases} \dfrac{\hat{c}_{ij} \left\lceil \frac{\underline{z}^\omega_{ij}}{Q} \right\rceil}{\underline{z}^\omega_{ij}}, & \text{if } \underline{z}^\omega_{ij} > 0, \\ \rho^t_{ij}, & \text{otherwise}, \end{cases} \tag{2.10}$$

where $\underline{z}^\omega_{ij}$ is the value of the continuous relaxation of the z variable for arc (i, j) in scenario ω from the solution of (2.9) in iteration t of the slope scaling procedure. The slope scaling portion of the algorithm is terminated when two successive slope scaling iterations yield design solutions that have approximately the same costs or if a time limit has been exhausted (implementation of this time limit is described in Section 2.6.3).

To initialize slope scaling in the very first iteration of SAA (i.e. when $m = 1$), we set $\rho^0_{ij\omega} = \frac{\hat{c_{ij}}}{Q} \quad \forall (i, j), \ \omega$. However, for subsequent iterations of SAA ($m > 1$), we make use of the multipliers found at the end of the slope scaling procedure in iteration $m-1$ of SAA. To do this, we can initialize the slope scaling multipliers as a convex combination of $\frac{\hat{c_{ij}}}{Q}$ (the starting multiplier) and iteration $m-1$'s final multipliers. The Heuristic-SS algorithm is outlined in Appendix A.1.4.

In summary, we use the following solution approaches: **Exact-Select**, **Exact-Enum**, **Exact-RandCut**, **Exact-RGContraction**, **Heuristic-Select**, **Heuristic-Enum**, **Heuristic-RandCut**, **Heuristic-RGContraction**, and **Heuristic-SS**.

We note that it is possible to combine static and dynamic generation of cut inequalities, i.e., enumerating cutsets of cardinality less than or equal a certain size up front, but dynamically "extending" some or all of these cutsets to larger ones that might be violated by a current solution. Computational experiments with such a combined approach revealed only minor improvements.

2.6 Computational Study

The primary objectives of this computational study are to:

1. demonstrate the effectiveness of $\mathbf{2}$-alt load plans in handling demand uncertainty by comparing the expected cost of operating with such load plans with their $\mathbf{1}$-alt and Infinite-alt counterparts,

2. gain insights into how $\mathbf{1}$-alt and $\mathbf{2}$-alt load plans differ,

3. assess the value of using stochastic models instead of deterministic models when solving the $\boldsymbol{p}$-alt model, and

4. select, from among the solution approaches presented in Section 2.5, an approach that increases our chances of finding close-to-optimal solutions to the stochastic $\boldsymbol{p}$-alt problem to facilitate the previous objectives.

This section is organized as follows. We first describe the instances and parameters used in our experiments in Section 2.6.1. In Section 2.6.2, we demonstrate the effectiveness of adding the cut inequalities (2.8), and compare the performance of the exact approaches. In Section 2.6.3, we compare the performance of the heuristic approaches. The objective of these two comparisons is to select a solution approach to use in the final set of experiments where we analyze $\mathbf{2}$-alt designs. Finally, in Section 2.6.4, we use the chosen solution approach to conduct the main experiment in this section. In this experiment, we demonstrate the effectiveness of $\mathbf{2}$-alt designs in handling demand uncertainty by comparing their total

expected costs with both $\mathbf{1}$-alt and Infinite-alt designs. The word *design* is used in this section to refer to a first-stage solution, i.e., decisions relating to the operation of scheduled trailers and the load plan structure. We also share some managerial insights related to $\mathbf{2}$-alt load plans.

All algorithms were implemented in Python. All experiments were run on an Intel i5 Quad-Core 2.6GHz computer with 8GB of RAM running Fedora 27, with Gurobi 7.5.2 used as the IP solver. In all cases, we set Gurobi's MIPFocus parameter to focusing more on finding feasible solutions.

2.6.1 Description of Instances and Parameters

In this study, instances are created using an instance generator which generates a graph representing the line-haul network of the carrier, as well as the OD demand information. To create a line-haul network, the generator randomly picks coordinates in the plane corresponding to EOL and BB terminals. Since the current technology for solving problems like the stochastic $\mathbf{p}$-alt problem is such that real-life instances are beyond our reach, our goal is to generate network structures with characteristics that resemble real LTL networks. To that end, we intervene manually and adjust the generated node/terminal coordinates whenever necessary to ensure that BB terminals are located more centrally in the network, and to ensure that the locations of EOL terminals are such that a reasonable network structure is generated. We allow three types of arcs in the graphs which are classified by the type of terminals they connect:

1. EOL-BB arcs: These are arcs that connect an EOL terminal to a BB terminal and represent movement of freight from an EOL to a BB. For all EOLs, we include in the graphs the arc connecting that EOL to its nearest BB terminal.

   In addition, we check two simple conditions to determine whether or not an EOL should have an additional connection to its *second* nearest BB terminal, i.e., whether or not an EOL is a multi-loading EOL. Specifically, for an EOL terminal, E, with

terminals B1 and B2 as its nearest and second nearest BBs, respectively, we check the following conditions: (1) if the distance of the direct connection E-B2 is less than 0.7 times the distance of transiting through the nearest BB terminal, i.e., E-B1-B2, and (2) if the distance of the direct connection E-B2 is less than 1.5 times the distance to the nearest BB terminal, i.e., E-B1. If both of these conditions are satisfied, then the arc representing the connection E-B2 is added to the graph. Using these conditions, only a small fraction of EOLs become multi-loading EOLs, as is the case in practice.

2. BB-EOL arcs: These are arcs that connect a BB terminal to an EOL terminal, and represent movement of freight from a BB to an EOL. For these arcs, we use the same setup as the one used for EOL-BB arcs described above, i.e., all EOL-BB arcs have BB-EOL counterparts included in the graph.

3. BB-BB arcs: These are arcs that connect a BB terminal to another BB terminal. All such arcs were included in our graphs.

The cost, c_{ij}, of an arc (i, j) in the graph was then set to be the Euclidean distance of that connection, and we set $\hat{c}_{ij} = 1.5\ c_{ij}$ for all arcs (i, j). Note that direct EOL-EOL connections are not included in our networks. This is justified as these connections rarely happen in practice, and even if they do, one can apply a pre-processing step that takes care of full (or nearly full) truckload shipments between these EOLs, and consider only the remaining partial truckload shipments in the model. An example of one of the networks generated and used in this study can be seen in Figure 2.5. This network is comprised of 12 EOL and 5 BB terminals. For completeness, we also include the expected demand data in an aggregated format in Table 2.3. For each EOL terminal, the 'Outgoing' column gives the sum of all the expected demand values (in trailer-loads) for the commodities that have that EOL as their origin, and the 'Destinations' column lists the EOL terminals that those commodities are destined for. Similarly, the 'Incoming' column shows the sum of

all the expected demand values (in trailer-loads) for the commodities that have that EOL as their destination, and the 'Origins' column lists the EOL terminals that those commodities originated from.

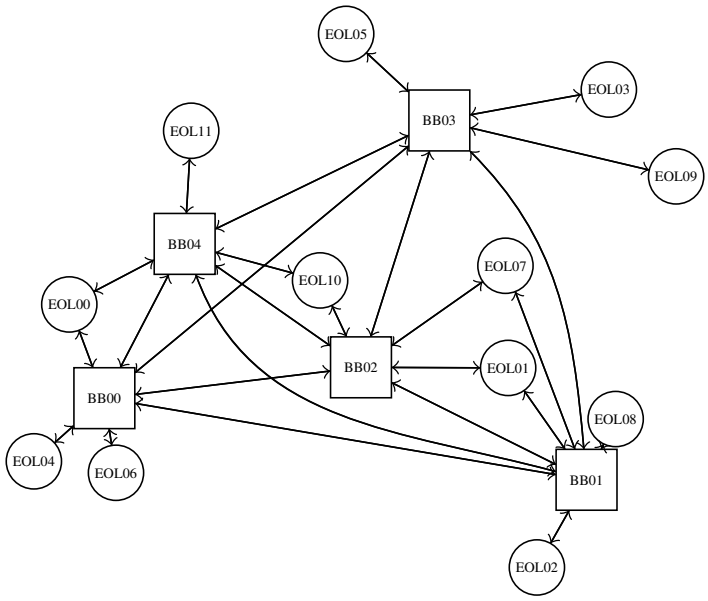


Figure 2.5: Example of LTL line-haul network generated by our instance generator; squares represent BB terminals while circles represent EOL connections

To create demand data, the generator takes as input the desired percentage of EOL pairs that make up the set of commodities $\mathcal{K}$ in the model, i.e., the EOL pairs that have positive demand (recall that only EOL terminals can act as origins/destinations in our setup). It translates the percentage into a *number* of EOL pairs, rounding up if necessary, and then randomly picks that many EOL pairs to represent the commodities in the model. Thus, all EOL pairs that are *not* selected as part of the set $\mathcal{K}$ have zero demand in every generated scenario. We assume that the demand for each of these commodities follows a truncated normal distribution with a user-defined upper and lower limit, both taken to be the same for all commodities. Expected demand values for each of the commodities are randomly generated within that interval. Furthermore, by specifying a uniform upper-limit on the standard deviation values, σ, standard deviations are randomly generated in the range $(0, \sigma)$ for each commodity. The user also supplies the value for the uniform trailer capacity, Q.

Table 2.3: Aggregated demand data for instance represented in Figure 2.5

EOL	Outgoing	Destinations	Incoming	Origins
0	0.21	3,9,10	2.78	1,2,3,4,10,11
1	1.50	0,3,9	1.46	2,7
2	2.26	0,1,3,6	1.42	3,8,10,11
3	3.52	0,2,4,7,9,10,11	0.69	0,1,2,6,9
4	2.62	0,5,6,9	2.78	3,6,9,11
5	0.26	8	3.15	4,6,8,9,11
6	2.58	3,4,5,7,8,10	2.44	2,4,7,9
7	1.28	1,6	0.85	3,6,8
8	1.12	2,5,7,9,10	1.29	5,6,9,10
9	2.25	3,4,5,6,8	2.49	0,1,3,4,8
10	1.14	0,2,8	1.51	0,3,6,8
11	2.58	0,2,4,5	0.47	3

For algorithm parameters, we set $M = 10$, $N = 10$, and $N' = 1000$ for the overarching SAA algorithm and set a time limit of 60 minutes for the solution of a sample problem (we experimented with other settings such as $M = 20$ and a time limit of 30 minutes, but results did not noticeably improve; see Appendix A.3). Our choices of SAA parameters yielded optimality gaps (according to Equation (2.5)) of less than 2% for all three models and all six instances used. For completeness, we include the optimality gap estimates for all instances in Appendix A.3.

For the cut generation algorithms described in Section 2.5.3, we set the following parameter values:

- ENUM: We set $M^{Enum} = 4$ (the size of cutsets we enumerate up front).
- RANDCUT: We set $RCMaxIter = 50$ and $\gamma^{RC} = 20$ for the root node, while we have $RCMaxIter = 15$ and $\gamma^{RC} = 10$ elsewhere.
- RGCONTRACTION: We set $RGMaxIter = 30$ and $\gamma^{RG} = 20$ for the root node, while we have $RGMaxIter = 15$ and $\gamma^{RG} = 10$ elsewhere. We also set $l = 3$ (the size of the RCL), and $\mathcal{N}^{final} = 3$.

We set $RGMaxIter$ to be lower than $RCMaxIter$ at the root node because its algo-

rithm is much slower in execution compared to RANDCUT. We also note that in the case of the heuristic methods, the callback functions containing the separation routines are executed after solving the linear programming relaxation in branch-and-bound nodes, as well as whenever a feasible solution to the relaxed model was found.

In all the experiments presented in this section, we solve all the evaluation subproblems in the SAA scheme exactly, using RANDCUT for generating cut inequalities (which was favored for its speed).

Table 2.4 provides the characteristics for the instances used in this study. The values of Q and the expected demand and standard deviation are all expressed in trailer-loads. The instances are labeled as follows: no. of EOLs - no. of BBs - % OD density - Q - σ. We note that the network in Figure 2.5 corresponds to the networks used in Instances 12-5-35-1-0.2 and 12-5-35-1-0.6. Similarly, the data in Table 2.3 corresponds to the same two instances (as both instances have the same expected demands).

Table 2.4: Instance characteristics

Instance	Line-haul Network			% OD density	No. of OD pairs $\lvert\mathcal{K}\rvert$	Q	Expected demand range	Demand std. dev. range
	No. of EOLs	No. of BBs	No. of arcs $\lvert\mathcal{A}\rvert$					
12-5-20-1-0.2	12	5	52	20	27	1	(0, 1)	(0, 0.2)
12-5-35-1-0.2	12	5	52	35	47	1	(0, 1)	(0, 0.2)
14-7-36-1-0.2	14	7	82	35	64	1	(0, 1)	(0, 0.2)
12-5-20-1-0.6	12	5	52	20	27	1	(0, 1)	(0, 0.6)
12-5-35-1-0.6	12	5	52	35	47	1	(0, 1)	(0, 0.6)
14-7-36-1-0.6	14	7	82	35	64	1	(0, 1)	(0, 0.6)

2.6.2 Comparison of Exact Approaches

The objective of this experiment is to compare the exact approaches Exact-Select (S), Exact-Enum (E), Exact-RandCut (RC), and Exact-RGContraction (RG). In addition, we include Exact-NoCuts (NC), where we solve the sample problems exactly but without the

cut inequalities, as a benchmark. Because we are primarily seeking good designs, we set the first stage variables to have a higher branching priority than their second stage counterparts. It is also important to note that the N-sample and the N'-sample generated for the i^{th} sample problem in SAA were the same across all the different methods we consider.

We analyze the results of solving instances 12-5-20-1-0.2, 12-5-35-1-0.2, and 14-7-35-1-0.2 using both **1**-alt and **2**-alt models. Although we only consider three instances, because we need to solve $M = 10$ sample problems for each combination of instance and model, this gives us a total of 60 sample problems to base our comparison upon. Each of the sample problems for a particular instance-model-method combination was allotted a computational time of one hour (for a combined total of 10 hours). This time does not include the evaluation phases which, in the worst case, may take up an additional total of 2-3 hours.

In Table 2.5, we report the following statistics about each instance and each method (each number in the table represents an average over $M = 10$ sample problems).

- IPGap: the average percentage optimality gap at termination as reported by the solver for the sample problems.
- GTB: the average gap relative to the best overall solution found by any of the five methods measured using the total expected cost (first-stage cost plus recourse cost estimate obtained using the $N' = 1000$ scenarios), calculated as $100 \times \frac{TotalCost - BestCost}{BestCost}$.
- TFD: the average time to final *design*, which for a given sample problem is defined as the time from the start of the solution process until the last observed change in the first-stage cost. Although the first-stage cost only reflects decisions about scheduled trailers and a design also includes decisions about the load plan (alt) structure, we use this cost as a proxy to check for design changes during the solve of a sample problem.
- No. B&B nodes: the average number of branch-and-bound nodes explored by the

solver during the sample problem solution process.

We also include a column containing the averages across all $\mathbf{1}$-alt and $\mathbf{2}$-alt instances (Averages), and we highlight the best values for that column in bold (except for the No. B&B nodes statistic).

From the IPGap values in Table 2.5, we readily observe that even for such relatively small networks with 17-21 terminals, these sample problems are difficult to solve to optimality. Among all 300 sample problems considered for all six combinations of instances and models, only five were solved to optimality. All of these five sample problems were in the $\mathbf{2}$-alt version of instance 12-5-20-1-0.2 and all were solved with the help of the cut inequalities. It is well-known that network design problems typically suffer from weak lower bounds which is a major contributor to their difficulty. We also observe that on average, the gaps at termination for the $\mathbf{2}$-alt models are less than their $\mathbf{1}$-alt counterparts. The RG method also stands out as the method achieving the best average IPGap on both $\mathbf{1}$-alt and $\mathbf{2}$-alt instances which suggests that this approach is an effective separation approach for the cut inequalities.

The best solution quality is consistently found by S on the $\mathbf{1}$-alt instances (GTB), while RG finds the best solutions, on average, in the $\mathbf{2}$-alt instances. Ultimately, there is a trade-off here between node exploration in the B&B search tree and spending time at each of those nodes separating for cut inequalities. For instance, by adding only some of the cut inequalities using knowledge of the instances, S purposefully spends less time looking for cut inequalities during the solution process, and is able to use that extra time to explore more of the search tree and find better solutions in case of the $\mathbf{1}$-alt instances. This is also corroborated by observing that the number of branch-and-bound nodes explored by S are much greater than that of RG and E. Furthermore, E adds the same cut inequalities that S adds (and many more), but this results in less exploration of the search tree, and an inability to match the solution quality found by S. In other words, adding too many cut inequalities can be a hindrance. To visually represent the trade-off between solution time and solution

Table 2.5: Statistics for exact methods

Method	Statistic	Instance						Averages	
		12-5-20-1-0.2		12-5-35-1-0.2		14-7-35-1-0.2			
		1-alt	**2**-alt	**1**-alt	**2**-alt	**1**-alt	**2**-alt	**1**-alt	**2**-alt
NoCuts (NC)	GTB (%)	0.00	0.03	0.19	0.03	0.21	0.51	0.13	0.19
	IPGap (%)	3.79	2.94	4.67	2.65	9.28	5.77	5.91	3.79
	TFD (s)	137.43	124.97	707.16	744.46	2,558.34	2,814.78	1,134.31	1,228.07
	No. B&B nodes	50,318.90	284,927.60	13,841.50	62,945.60	792.50	4,956.10	21,650.97	117,609.77
RGContraction (RG)	GTB (%)	0.00	0.00	0.01	0.03	0.33	0.00	0.12	**0.01**
	IPGap (%)	3.37	0.26	4.67	1.20	8.43	4.18	**5.49**	**1.88**
	TFD (s)	1,066.15	263.05	1,408.66	1,518.35	2,654.49	2,362.55	1,709.77	1,381.32
	No. B&B nodes	4,518.00	7,353.90	3,894.60	6,078.60	509.50	1,044.90	2,974.03	4,825.80
RandCut (RC)	GTB (%)	0.00	0.08	0.09	0.00	0.47	0.28	0.19	0.12
	IPGap (%)	3.11	0.30	4.34	1.60	9.30	5.40	5.58	2.43
	TFD (s)	303.67	140.84	535.58	873.63	2,153.10	2,623.25	**997.45**	1,212.57
	No. B&B nodes	34,670.40	173,433.00	10,956.80	53,883.10	763.30	4,237.10	15,463.50	77,184.40
Enum (E)	GTB (%)	0.54	0.19	0.20	0.18	1.61	0.64	0.78	0.34
	IPGap (%)	4.33	3.18	4.61	1.96	9.87	5.87	6.27	3.67
	TFD (s)	346.77	125.34	1,109.39	796.11	1,968.85	2,302.50	1,141.67	1,074.65
	No. B&B nodes	24,612.80	203,683.10	7,126.20	66,492.90	70.40	830.40	10,603.13	90,335.47
Select (S)	GTB (%)	0.00	0.06	0.00	0.07	0.00	0.54	**0.00**	0.22
	IPGap (%)	3.40	2.10	4.49	1.67	9.64	5.61	5.84	3.13
	TFD (s)	390.07	121.63	786.52	451.99	2,578.17	1,862.35	1,251.59	**811.99**
	No. B&B nodes	45,629.30	361,577.50	12,256.70	136,932.60	748.80	3,729.30	19,544.93	167,413.13

quality, we include in Appendix A.2 plots showing this trade-off (Figure A.1).

We also observe a trade-off between a design's quality and when that design is found in the solution process, represented by the TFD statistic. On average, RG takes the most time to find its final design on both sets of instances, while RC and S are the quickest on the $\mathbf{1}$-alt and $\mathbf{2}$-alt instances, respectively. A visual representation of this trade-off can be seen in Figure A.2 in Appendix A.2.

Although there does not seem to be an exact method that clearly and consistently dominates on all metrics, S and RG outperform the other three. While we could choose either of those two as the best exact approach, we ultimately favor S due to its simplicity and its lower TFD values on both sets of instances.

2.6.3 Comparison of Heuristic Approaches

The objective of this experiment is to compare the heuristic approaches Heuristic-Select (S), Heuristic-Enum (E), Heuristic-RandCut (RC), Heuristic-RGContraction (RG), and Heuristic-SS (SS). In addition, we include Heuristic-NoCuts (NC), where we solve the relaxed sample problems but without making use of the cut inequalities, as a benchmark. Again, both the N-sample and the N'-samples generated for the i^{th} sample problem were the same across all the different heuristic methods.

For NC, RC, E, and RG, we set a time limit of an hour for solving a sample problems. For SS, to solve a sample problem, we may have to perform several iterations of slope scaling. In each of these iterations, we solve a MIP given by (2.9), referred to as SSMIP, with a certain set of multipliers before adjusting the multipliers again using (2.10) until a stopping criterion is met. For this reason, we implement a timing scheme for SS that takes this into consideration. Specifically, this scheme consists of two components:

1. The *total* time limit for solving the SSMIPs of a single sample problem, set to one hour.

   We observed that the first few SSMIPs solve in little time, but subsequent SSMIPs are

generally more difficult to solve, thereby causing SS to exhaust the total time limit having performed only a few slope scaling updates. However, we also observed that in most cases a high-quality (or optimal) solution to each SSMIP is found early in the solution process, and that most of the time spent on the harder SSMIPs is likely spent proving optimality. Therefore, we implemented an individual time limit for each SSMIP.

2. An *individual* time limit for each SSMIP, determined dynamically.

   Experiments with SS on smaller instances suggested that the average number of slope scaling iterations taken by a sample problem was around five. Therefore, we allocate the one hour among the individual SSMIPs as follows: the first SSMIP is allowed 12 minutes, the time taken by the first and second SSMIPs combined is not to exceed 24 minutes, and so on. Therefore, if a SSMIP of a sample problem finishes before exhausting its individual time limit, the remaining time is carried over to the next SSMIP for that sample problem.

Additionally, in Heuristic-SS, to solve sample problem $m > 1$ in the SAA process, we initialize the slope scaling multipliers for iteration m using: $\rho^m_{ij\omega} = 0.7\rho^{m-1}_{ij\omega} + 0.3\frac{\hat{c}_{ij}}{Q}\ \forall i, j, \omega$, where $\rho^{m-1}_{ij\omega}$ represents the final multipliers at the final SSMIP of sample problem $m-1$ for arc (i, j) and scenario ω.

Again, we analyze the results of solving instances 12-5-20-1-0.2, 12-5-35-1-0.2, and 14-7-35-1-0.2 using both **1**-alt and **2**-alt models. In Table 2.6, we show the results of these runs. In addition to GTB and TFD which were previously defined, we define the statistic RCP as the average (over 10 sample problems) percentage of the expected total cost that is comprised by the recourse cost obtained from the evaluation phase with 1,000 scenarios. In the Averages column in the table, we highlight the best values for GTB and TFD in bold. Furthermore, we refer the reader to Figures A.1 and A.2 in Appendix A.2 for plots of the expected total cost vs. solution time, expected total cost vs. TFD, and a more in-depth

discussion of the results.

We readily observe from Table 2.6 that SS consistently finds the best solution in all six instances. The next best, in terms of solution quality, is RG, while the other three approaches find solutions that are much worse compared to SS and RG. While RC is the fastest in finding a design, the quality is not much better than not adding any cut inequalities. An effective separation routine is critical for the heuristics that use the cut inequalities, as we need to increase the likelihood of cutting off solutions found by the solver to the relaxed problem by finding violated cut inequalities. We see that RG does a much better job than RC generating effective cuts and mitigating the effects of relaxing the second-stage variables.

Looking at the RCP values in Table 2.6, we can also make the observation that methods that find solutions with higher quality (e.g. SS) exhibit a lower RCP value, i.e., the percentage of the expected total cost that is made up by the recourse cost is very low. This is natural as relaxing the sample problems leads to an underestimation of the recourse cost in those problems, thereby favoring meeting more demand than usual using the second-stage additional capacity rather than the first-stage capacity, which, in turn, results in low-quality first-stage decisions. This observation is consistent with what we observed in solutions obtained via the exact methods (in which RCP ranges from 3% to 7%).

Because SS consistently finds the design with the best quality, we select SS as the best among the heuristic approaches.

2.6.4 Analysis of $\mathbf{2}$-alt Designs

The primary objective of our experiments is to analyze $\mathbf{2}$-alt designs and highlight their potential. We accomplish this by comparing $\mathbf{2}$-alt designs to their $\mathbf{1}$-alt and Infinite-alt counterparts using a number of different metrics. A secondary objective of our experiments is to demonstrate the value of obtaining a design by using stochastic models rather than deterministic ones.

Table 2.6: Statistics for heuristic methods

Method	Statistic	Instances						Averages	
		12-5-20-1-0.2		12-5-35-1-0.2		14-7-35-1-0.2			
		1-alt	**2**-alt	**1**-alt	**2**-alt	**1**-alt	**2**-alt	**1**-alt	**2**-alt
NoCuts (NC)	GTB (%)	17.24	10.69	27.17	11.67	18.31	6.29	20.91	9.55
	TFD (s)	5.66	6.36	21.32	50.16	1531.12	563.64	519.37	206.72
	RCP (%)	43.90	34.60	49.50	36.80	40.70	28.30	44.70	33.23
RGContraction (RG)	GTB (%)	3.13	0.97	6.25	0.57	9.84	0.02	6.41	0.52
	TFD (s)	183.04	61.50	800.57	247.55	1879.59	1736.04	954.40	681.70
	RCP (%)	14.40	6.20	15.70	6.10	25.50	9.50	18.53	7.27
RandCut (RC)	GTB (%)	14.90	9.79	24.40	11.74	18.53	6.57	19.28	9.37
	TFD (s)	13.11	8.49	53.05	9.17	247.05	353.38	**104.40**	**123.68**
	RCP (%)	39.00	31.90	46.80	37.60	40.90	27.90	42.23	32.47
Enum (E)	GTB (%)	12.57	6.95	14.80	5.29	16.78	4.64	14.72	5.62
	TFD (s)	46.72	15.03	154.13	68.64	1419.65	1114.34	540.16	399.33
	RCP (%)	32.57	24.98	32.07	19.94	36.18	23.28	33.61	22.73
Select (S)	GTB (%)	16.86	9.73	15.28	8.98	17.57	6.30	16.57	8.34
	TFD (s)	25.52	10.20	115.72	22.86	999.41	1110.21	380.22	381.09
	RCP (%)	40.07	33.14	37.58	31.47	39.22	27.30	38.95	30.64
Slope Scaling (SS)	GTB (%)	0.00	0.00	0.00	0.00	0.00	0.00	**0.00**	**0.00**
	TFD (s)	1865.30	480.33	2552.28	1513.04	3175.66	3453.83	2531.08	1815.73
	RCP (%)	6.70	3.90	4.10	3.00	5.70	4.40	5.50	3.77

For the next experiment, and to facilitate making direct comparisons between designs, we generate 1,000 scenarios up front to represent all possible realizations of demand that can occur, and use these 1,000 scenarios exclusively for all the runs of the experiment. In particular, the $N = 10$ scenarios for the sample problems are now randomly sampled from these 1,000 scenarios, and all resulting designs are evaluated on the *same* 1,000 scenarios (i.e. evaluation is exact). Furthermore, we favored Exact-Select for this experiment over Heuristic-SS due its slight edge in terms of finding better designs within the time limit (Figure A.1 in Appendix A.2). The results reported are based on the solutions to our six instances (see Table 2.4 for instances characteristics).

The benefits of $\mathbf{2}$*-alt designs*

In this section, we compare $\mathbf{1}$-alt, $\mathbf{2}$-alt, and Infinite-alt designs on a number of metrics to demonstrate the benefits that can be realized by adopting $\mathbf{2}$-alt designs. For each model and instance, we compute the following metrics associated with the *chosen* design:

1. *Cost*: This is the primary metric that we use to compare designs. The costs reported here for a design are the total expected costs, i.e., the cost of the first-stage decisions plus the expected recourse cost when evaluated over the 1,000 scenarios.

2. *Consolidation metrics*: LTL carriers are consolidation carriers, and hence, improving the consolidation of commodities/shipments naturally results in cost savings. We use consolidation metrics, therefore, to shed some light on where cost savings come from. As it is difficult to define a single consolidation metric that accurately captures the level of consolidation provided by a design, proxies are used (see [4] and [49] for similar approaches). Proxies generally fall in one of two categories: (1) metrics describing the level of *multipath usage* and (2) metrics describing the level of *path sharing*. Generally speaking, it is expected that if more commodities have multiple paths to get from their origins to their destinations, and if more commodities share arcs in their paths from their origins to their destinations, the higher the likelihood

that freight is indeed consolidated on the services of the network. We use the following three metrics:

(a) *# Commodities Consolidated*: For each design, we report the average number of commodities that are consolidated on an arc (taken across all 1,000 scenarios). This is done by computing, for each scenario, the number of commodities whose flow variables are positive on a particular arc, then averaging those values over the set of 1,000 scenarios, and then again over the set of all arcs in the network.

(b) *# OD Paths*: For each design, we report the average number of paths in the load plan between origins and destinations. This is done by computing, for each commodity, the number of paths in the load plan that can be used to get from its origin to its destination. Since alt variables have no cost, we are careful here to only include in the load plans the alts selected by the solver if at least one commodity in a load plan uses that arc. Then, we average these numbers over the set of all commodities.

(c) *Utilization*: For each design, we report here the average percent utilization of trailers taken over all 1,000 scenarios. For each scenario ω, we calculate its average percent utilization as follows:

$$u_{scenario} = 100\left(\frac{1}{|\mathcal{A}^{used}|}\sum_{(i,j)\in\mathcal{A}^{used}}\frac{\sum_{k\in\mathcal{K}} x^{\omega}_{ijk}}{r_{ij}+z^{\omega}_{ij}}\right), \tag{2.11}$$

where $\mathcal{A}^{used}$ here represents that set of arcs on which at least one trailer (regardless of its type) is operated. These values are then averaged over all 1,000 scenarios.

3. % Alt Subset: We use this metric specifically to compare $\mathbf{1}$-alt and $\mathbf{2}$-alt load plans. This metric is defined as the average percentage of arcs in the $\mathbf{1}$-alt load plan that are

also part of the **2**-alt load plan. We compute this percentage for each destination in the network, and then average the percentages across all destinations to arrive at the overall average percentage.

Table 2.7: Cost results

Instance	Model			Savings (%)	Gap in Savings (%)	Gap closed (%)
	1-alt	**2**-alt	Infinite-alt			
12-5-20-1-0.2	849.47	798.97	797.37	5.94	6.13	96.90
12-5-20-1-0.6	879.43	823.48	822.52	6.36	6.47	98.30
12-5-35-1-0.2	1243.57	1174.60	1172.78	5.55	5.69	97.54
12-5-35-1-0.6	1290.88	1215.10	1208.78	5.87	6.36	92.30
14-7-35-1-0.2	1745.59	1605.16	1593.53	8.04	8.71	92.31
14-7-35-1-0.6	1814.37	1658.49	1651.27	8.59	8.99	95.55
			Average	6.73	7.06	95.48

Table 2.7 shows the total expected costs for all three models and all six instances. We use the Infinite-alt model here as a benchmark since it completely relaxes the restrictive load plan requirements. The "Savings" column represents the percentage savings obtained from using a **2**-alt design over a **1**-alt design. The "Gap in Savings" column presents the percentage savings obtained from using an Infinite-alt design over a **1**-alt design, i.e., this represents the gap that is available between the most restrictive design (**1**-alt) and the least restrictive one (Infinite-alt). The "Gap closed" column shows the percentage of the gap between the **1**-alt and Infinite-alt costs that the **2**-alt model is able to close. Specifically, we calculate the gap closed as the ratio of the "Savings" column to the "Gap in Savings" column multiplied by 100.

The first observation we make is that, for these instances, adopting a **2**-alt design results in cost savings of almost 7%. In fact, for the largest two instances, the savings are more than 8%. This demonstrates the inherent restrictiveness of the **1**-alt load plans, and that injecting a little bit of additional flexibility by adding a second alt is enough to generate substantial cost savings. For carriers, reducing their costs by 6-7% is a significant amount.

For instance, for a large US LTL carrier, cost savings in the order of 6-7% translate roughly into $300,000 per week in savings [41]. Furthermore, we note that all instances with higher variability (instances whose identifiers end in 0.6) exhibit higher savings than their low variability counterparts. This suggests that adopting **2**-alt load plans in more uncertain environments is highly beneficial, and supports what these carriers are actually doing in practice by adding a second alt to some terminal-destination pairs. Of course, in practice, these carriers often solve deterministic models instead of stochastic ones to obtain **1**-alt designs, but we still show that the practice of adding alts to cope with demand variations is, at the very least, a good idea.

A natural question to ask then is: "How much would the carrier save if it were to add a third alt to its load plans, then a fourth one, and so on?". We answer this by benchmarking the **2**-alt models against the Infinite-alt models and observing the Gap closed column in Table 2.7. On average, adopting a **2**-alt design closes 95.48% of the cost gap between the **1**-alt and Infinite-alt designs which represent the two extremes. In other words, just adding a second alt to a traditional load plan yields cost savings that are very close to allowing all alts. While this result might be an artifact of the sizes and structure of the instances we use, we still think the benefits would exist if larger networks are studied (even though the percentage of the cost gap closed may be different from what we found). In fact, our findings are also consistent with what was observed in [40] where the authors studied unrestricted load plans (these correspond to our Infinite-alt models) for real-life instances and gathered statistics about how many outbound arcs were utilized for a terminal-destination pair. They report that the vast majority (roughly 90%) of terminal-destination pairs use only 1-2 outbound arcs, which is exactly what our **2**-alt model captures.

This is an example of a situation where allowing a system some limited flexibility gives rise to benefits that are close to allowing full flexibility in the system. Another, famous example, is the "long-chain" concept presented in [55] in the context of manufacturing flexibility. These surprising benefits of limited flexibility come as good news for carriers,

since operating a full-flexibility load plan involves complex workflows and requires a high-level of automation. Carriers prefer traditional load plans for their operational simplicity, and what the results of these experiments show is that with only two alts allowed for each terminal and ultimate destination, the carrier can reap most of the benefits of full flexibility with only a fraction of the complexity.

Table 2.8: Comparison of design costs

Instance	Design Costs			DCP		
	1-alt	**2**-alt	Infinite-alt	**1**-alt	**2**-alt	Infinite-alt
12-5-20-1-0.2	800.31	764.54	762.89	94.21	95.69	95.68
12-5-20-1-0.6	800.31	761.92	761.92	91.00	92.52	92.63
12-5-35-1-0.2	1219.36	1137.07	1140.89	98.05	96.80	97.28
12-5-35-1-0.6	1200.12	1120.38	1108.63	92.97	92.20	91.71
14-7-35-1-0.2	1668.25	1505.16	1513.49	95.57	93.77	94.98
14-7-35-1-0.6	1688.53	1483.78	1489.46	93.06	89.47	90.20
Average	1229.48	1128.81	1129.55	94.15	93.41	93.75

In Table 2.8, we show the *design* costs, i.e., the first-stage costs, for all pairs of instances and models. We also report the Design Cost Percentage (DCP), defined as the percentage of the total expected cost that is made up of the design cost. When comparing the design costs and DCP values of the different models, it appears that, on average, the **2**-alt and Infinite-alt designs are comparable, while the **1**-alt designs have higher design costs and DCP values. Therefore, we observe that **1**-alt designs tend to invest more in scheduled trailers, whereas **2**-alt and Infinite-alt designs rely more on outsourced trailers. This is, at least in part, due to the increase in consolidation options in the case of **2**-alt load plans. This observation can potentially be exploited in designing a heuristic for the **2**-alt model.

To shed some light on where the **2**-alt cost savings come from, we report in Table 2.9, the consolidation metrics described above for all three models and all six instances. Because these statistics can potentially be distorted by the presence of a large number of alternative optimal solutions, we implement a hierarchical multi-objective scheme in each of the evaluation subproblems to try and reduce the impact that this might have. Specifically,

Table 2.9: Consolidation statistics

Instance	# Comms. Consolidated			Utilization (%)			# OD Paths		
	1-alt	**2**-alt	Infinite-alt	**1**-alt	**2**-alt	Infinite-alt	**1**-alt	**2**-alt	Infinite-alt
12-5-20-1-0.2	1.63	1.85	2.18	71.78	77.34	79.33	1.00	2.93	11.07
12-5-20-1-0.6	1.63	1.89	2.09	70.23	76.32	74.97	1.00	3.37	13.44
12-5-35-1-0.2	2.71	3.01	3.61	73.25	78.26	77.95	1.00	2.81	17.26
12-5-35-1-0.6	2.73	3.09	3.49	71.16	78.06	76.62	1.00	3.23	19.96
14-7-35-1-0.2	2.46	2.84	2.71	75.97	83.03	83.75	1.00	4.58	47.00
14-7-35-1-0.6	2.50	2.84	2.70	74.94	82.93	81.47	1.00	4.95	139.44
Average	2.28	2.59	2.79	72.89	79.32	79.02	1.00	3.65	41.36

we initially solve the evaluation subproblem as described above and obtain its optimal total cost, z^*, and a vector of flow variables, $\mathbf{x}$. Next, using the vector $\mathbf{x}$ and for each commodity, we define a subgraph consisting of all the arcs that had positive flow for that commodity in this initial solution, and use that to define path-flow variables for all paths in this subgraph that connect its origin to its destination. We also define binary variables that indicate if this path is used or not. Then, using these variables, we re-solve the model but now seeking a solution that minimizes the total number of paths that are used by all commodities subject to maintaining the value of z^* found in the initial solution.

On average, the **2**-alt designs consolidate a higher number of commodities and have a higher utilization than the **1**-alt designs, and the values are closer to those of the Infinite-alt designs than to those of the **1**-alt designs. Another interesting observation is that the number of OD paths that a commodity can follow from origin to destination in the **2**-alt case is an order of magnitude smaller than that of the Infinite-alt case. While we expect the number of paths in the **2**-alt case to be more than one, the increase is small with only 2-3 more paths for a commodity on average. Furthermore, we note that instances with higher variability allow for the use of more OD paths per commodity in both the **2**-alt and Infinite-alt model case, and therefore, the usage of multiple paths for each commodity appears to be a natural way to protect against demand volatility. A concrete example of this difference can be seen in the example shown in Figure 2.6, where we show the load

plans for both the $\mathbf{1}$-alt and $\mathbf{2}$-alt models for destination terminal EOL05 in the instance of Figure 2.5. In this example, only terminals EOL04, EOL06, EOL08, EOL09, and EOL11 have freight that is destined for EOL05, and for the purpose of this discussion, we will refer to commodities by their origin terminals. In Figure 2.6, we observe the increase in the number of OD paths available for each of the commodities in the $\mathbf{2}$-alt load plan compared to its $\mathbf{1}$-alt counterpart. Whereas the $\mathbf{1}$-alt load plan only allows a single path for each OD pair, the $\mathbf{2}$-alt load plan takes advantage of the additional routing options to provide some additional flexibility. This allows for freight to be "shuffled around" the network to avoid congested links in certain demand realizations, thereby utilizing existing capacity that may be available elsewhere. For instance, if the scheduled capacity on the link (BB00, BB03) is used up in its entirety on a certain day (could even be used by other commodities not destined for EOL05), then freight EOL04 and EOL06 can be routed along the links (BB00,BB02) followed by (BB02, BB03), and, therefore, bypass the congested link.

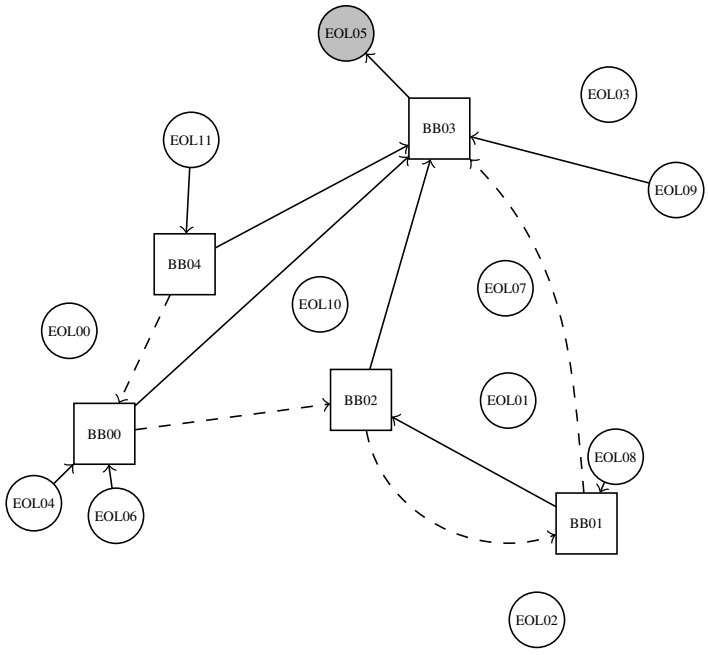


Figure 2.6: Example of load plan structures for the destination terminal EOL05 in Instance 12-5-35-1-0.2; the solid arcs represent the $\mathbf{1}$-alt load plan, and the dashed arcs together with the solid arcs represent the $\mathbf{2}$-alt load plan

In our experiments, we set the ratio of the cost of outsourced trailers to that of scheduled

Table 2.10: Analysis of the ratio of cost of outsourced trailers to scheduled trailers

Ratio	Average Total Expected Cost			Average DCP		
	1-alt	**2**-alt	Infinite-alt	**1**-alt	**2**-alt	Infinite-alt
1.05	1261.72	1170.13	1162.50	85.51	80.94	80.01
1.15	1277.80	1187.29	1180.62	89.92	88.19	87.96
1.25	1288.50	1196.57	1192.04	91.39	89.56	90.03
1.50	1303.89	1212.63	1207.71	94.15	93.41	93.75
1.75	1316.19	1224.57	1218.66	93.96	93.08	93.73
2.00	1325.83	1236.43	1234.87	93.56	93.33	93.25

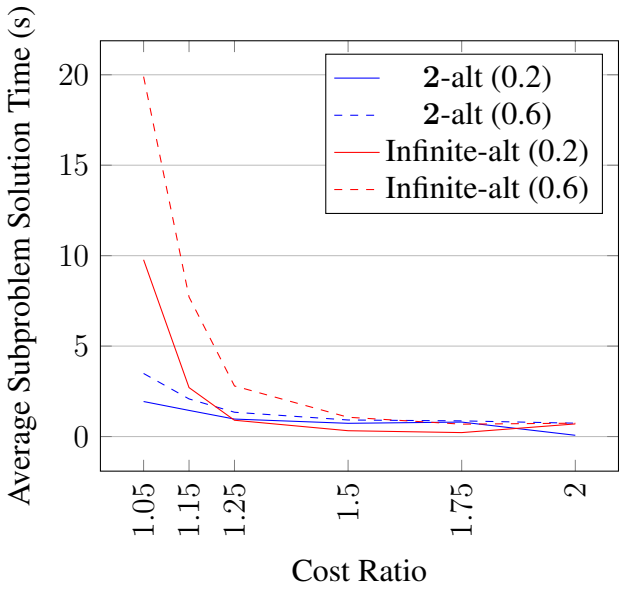


Figure 2.7: Analysis of evaluation subproblem solution time with respect to the ratio of the cost of outsourced trailers to scheduled trailers for instances 14-7-35-1-0.2 (solid) and 14-7-35-1-0.6 (dashed)

trailers to 1.5. In Table 2.10, we compare different values of this ratio and, for each, we summarize the results of the average total expected cost and the average DCP values (i.e., all numbers are averaged over all six instances). We consider ratios in the range from 1.05 to 2. The higher end of this range is of interest when the cost of an outsourced trailer is interpreted as a price quoted by the owner-operator to the LTL carrier. Such a price may be high relative to the carrier's own operating cost, even when the owner-operator's profit margin is small in absolute terms. Furthermore, the cost of an outsourced trailer can be used to model more than just the monetary cost. For example, it may be inflated to reflect the risk of not having that recourse available on short notice. Smaller ratios may be useful to model the situation in which a carrier resorts to overtime pay to have its own drivers operate "outsourced" trailers. In that case, to still operate profitably, the cost ratio of outsourced trailers to scheduled trailers should be small. This interpretation is, in fact, the one chosen in [10]. We observe that while the total expected cost increases steadily as the ratio is increased (as expected), the DCP value increases noticeably when increasing the ratio from 1.05 through 1.5, but then remains fairly stable after that: as expected, using smaller cost ratios produces less conservative designs as the model is more reliant on outsourced trailers due to them being relatively cheap. With larger cost ratios (e.g., 1.75 and 2), however, the solution shifts to relying more on scheduled trailers because outsourced trailers are now relatively more expensive. As the ratio increases, the $\mathbf{2}$-alt model is increasingly helpful in achieving lower total expected costs relative to that achieved with the Infinite-alt model: the gap between the two decreases from 0.66% when the ratio is 1.05 to only 0.13% when the ratio is 2.

Solution times are also affected by the ratio. The average solution time for the evaluation subproblems is noticeably increased when moving to smaller ratios. This is particularly true for the Infinite-alt runs and is most visible on the largest two instances. Figure 2.7 shows the average solution time for the evaluation subproblems for the instances 14-7-35-1-0.2 and 14-7-35-1-0.6 for the $\mathbf{2}$-alt and Infinite-alt runs (the $\mathbf{1}$-alt run did not exhibit a

noticeable increase as the ratio was decreased). As smaller ratios yield designs that are increasingly reliant on outsourced trailers, more work is needed by the evaluation sub-problems to decide where to add outsourced capacity. Since the Infinite-alt model allows more options, this greatly increases solution times when compared with the $\mathbf{1}$-alt and $\mathbf{2}$-alt models.

Table 2.11: Percentage of arcs of $\mathbf{1}$-alt load plan that are in $\mathbf{2}$-alt load plan

Instance	% Alt Subset
12-5-20-1-0.2	98.57
12-5-20-1-0.6	95.83
12-5-35-1-0.2	92.15
12-5-35-1-0.6	97.48
14-7-35-1-0.2	89.90
14-7-35-1-0.6	100.00
Average	95.66

Finally, in Table 2.11, we compare the percentages of arcs in the $\mathbf{1}$-alt chosen design (averaged over all destinations) with those of their $\mathbf{2}$-alt counterparts, % Alt Subset. On average, a high percentage of the arcs that were chosen by the $\mathbf{1}$-alt model are chosen again as one of the two alts in the $\mathbf{2}$-alt model (approx. 96%). For example, the load plan for EOL05 shown in Figure 2.6 shows a case where all the $\mathbf{1}$-alt load plan arcs also appear in the $\mathbf{2}$-alt load plan for that terminal.

Since there may well be multiple optimal solutions with different choices for the alt variables, these percentages are not definitive (and could even be higher). However, this suggests that (good) $\mathbf{1}$-alt designs have the potential to be reasonable starting points and can be extended to good $\mathbf{2}$-alt designs. This information, combined with the observations above, can be used to design heuristics for the $\mathbf{2}$-alt model.

Overall, these results indicate that $\mathbf{2}$-alt designs are an attractive option when it comes to designing more flexible load plans. These designs capture most of the cost/consolidation benefits of the fully-flexible Infinite-alt load plans without the added complexity associated with operating such designs. Furthermore, we presented a number of observations that

offer some managerial insights into why $\mathbf{2}$-alt designs performs so well, and how these designs harness these benefits.

The value of stochastic models

In this section, we present results that compare the use of a deterministic $\boldsymbol{p}$-alt model (with expected demand values as input) to the use of a stochastic $\boldsymbol{p}$-alt model. In the motivating example presented in Section 2.2.3, we presented a small example of an instance where solving a stochastic version of both the $\mathbf{1}$-alt and the $\mathbf{2}$-alt models provides better solutions than their deterministic equivalents. In the next experiment, we will solve a deterministic model using the expected demand for both the $\mathbf{1}$-alt and $\mathbf{2}$-alt models. Solving such a model is equivalent to solving a single sample problem (2.3) with a single scenario corresponding to the expected demand values. We again use the Exact-Select algorithm to solve this model, and we limit the solver to an hour of computational time. The design obtained is evaluated on the same 1,000 scenarios that were used to evaluate the stochastic model, and a total expected cost is calculated.

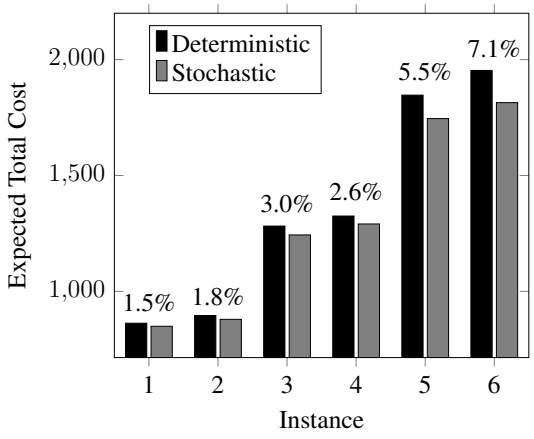


Figure 2.8: Expected total cost comparison between the stochastic and deterministic $\mathbf{1}$-alt designs; percentage *savings* are labeled on the plot and instance number corresponds to the order of Table 2.4

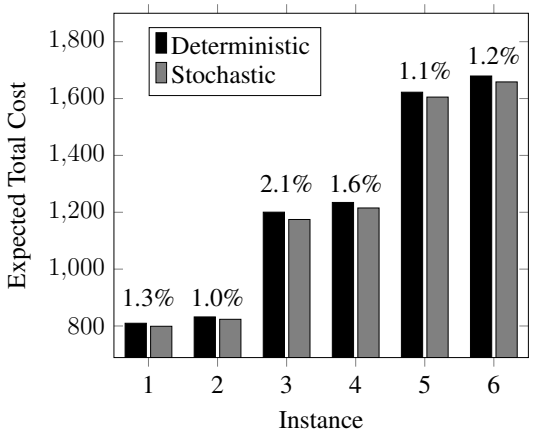


Figure 2.9: Expected total cost comparison between the stochastic and deterministic $\mathbf{2}$-alt designs; percentage *savings* are labeled on the plot and instance number corresponds to the order of Table 2.4

In Figure 2.8, we compare the total expected costs of the $\mathbf{1}$-alt deterministic and stochas-

tic designs. On average, across the six instances, the stochastic $\mathbf{1}$-alt model yields a 3.6% savings over the deterministic model. The savings are more pronounced on the larger instances which yield a considerable 5-7% savings. In Figure 2.9, we compare the total expected costs of the $\mathbf{2}$-alt deterministic and stochastic designs. On average, across the six instances, the stochastic $\mathbf{2}$-alt model yields a 1.4% savings over the deterministic model - much less than that of the $\mathbf{1}$-alt model. This suggests that, at least on these instances, a $\mathbf{2}$-alt deterministic model might be a good starting point for a heuristic for the stochastic $\mathbf{2}$-alt model, and may even be used in lieu of solving a stochastic program as it provided good solutions compared to the best solutions found using the stochastic model. We note, however, that despite the deterministic model containing a single scenario, we were only able to solve instances 1-4 to optimality within the one hour time limit. The computational time limit was exhausted by the two largest instances, and the optimality gap reported at termination for those two instances was 4-5%.

Therefore, these instances show that carriers can generate considerable savings by moving from solving a deterministic $\mathbf{1}$-alt model to a stochastic one, and then even more savings by allowing for two alts and solving a $\mathbf{2}$-alt stochastic model (for a combined savings of about 10%). We note that most of the models in the literature that have studied $\mathbf{1}$-alt (traditional) load plans for LTL carriers have been deterministic, so our findings here show the value of explicitly considering uncertainty in designing load plans for these carriers. The results also indicate that there may be potential for deterministic $\mathbf{2}$-alt models to be reasonable heuristics for stochastic models, or a good starting point for one.

2.7 Conclusion

We have introduced the $\boldsymbol{p}$-alt model, a service network design model that enables LTL carriers to extend their traditional load plans by allowing for additional flexibility when faced with uncertain demands. While traditional load plans allow for only one option (or "alt") for routing freight for each terminal-destination pair in the network, the $\boldsymbol{p}$-alt

model allows for up to p_{id} options for each pair of terminal i and destination d. We have investigated a number of exact and heuristic approaches for solving the two-stage stochastic $\boldsymbol{p}$-alt problem within the context of a Sample Average Approximation framework. The Exact-Select method has been chosen to carry out the final analysis of alts where we have studied the benefits of adding a single additional alt for terminal-destination pairs in the network, and compared the benefits of that limited form of flexibility with the benefits of allowing all options for each terminal-destination pair.

On the instances used, we have shown that adopting a design generated by a stochastic $\mathbf{2}$-alt model yields 6-7% savings compared to its $\mathbf{1}$-alt counterpart. Not only are the savings considerable, but just adding a single additional option at terminal-destination pairs closes about 95% of the cost gap between the $\mathbf{1}$-alt model and the Infinite-alt model (which completely relaxes the load plan requirements). That is, the $\mathbf{2}$-alt load plan designs offer benefits that are close to that of the Infinite-alt model, while maintaining a good degree of the operational simplicity that is characteristic of the $\mathbf{1}$-alt load plan. These cost savings are generated by the added consolidation opportunities created by the presence of these second alts (indicated by an increase in the average number of commodities consolidated per arc, an increase in the number of OD paths per commodity, and an increased utilization rate).

We also show the value of using a stochastic $\boldsymbol{p}$-alt model compared to a deterministic one. Our findings here indicate that adopting a stochastic $\mathbf{1}$-alt model yields average savings in the order of 3-4% over a deterministic one. As many carriers currently adopt deterministic $\mathbf{1}$-alt models, switching to stochastic $\mathbf{2}$-alt models, therefore, yields a combined savings in the order of about 10%.

There are many future research directions which can be pursued to extend this work, primarily those related to solution approaches for the $\boldsymbol{p}$-alt model. While we investigated a number of heuristic approaches to solve the stochastic $\boldsymbol{p}$-alt model, we retained the integrality of the first-stage variables in doing so. Due to the difficulty of solving the stochastic $\boldsymbol{p}$-alt model, heuristics that tackle the stochastic program directly are worth exploring. The

managerial insights we share in this chapter can be used as a starting point for developing such heuristics. Another potential research direction is studying the polyhedral structure of the p-alt problem. Different formulations of this problem can be compared against one another, and cuts that take into consideration the inherent structure of the problem may be identified.

Part II

Operational Planning

CHAPTER 3
THE DYNAMIC FREIGHT ROUTING PROBLEM FOR LTL CARRIERS USING APPROXIMATE DYNAMIC PROGRAMMING

3.1 Introduction

As discussed previously, much of the literature on LTL carriers focuses on tactical planning – generally referred to as *service network design*. In Chapter 2, we show that to accommodate inherent demand uncertainty, relaxing the traditional 1-alt load plan requirements by allowing two next terminal options for location-destination pairs in a 2-alt load plan can lead to substantial savings. Furthermore, these savings are comparable to the savings generated when operating with a load plan in which, for each location-destination pair, all terminals can be next terminal options – a plan which is much more complex to manage operationally.

The work in this part of the thesis is motivated by these findings and focuses on how to effectively operate a 2-alt load plan. Although the ideas explored here can be applied to a more general load plan structure, we restrict our attention to 2-alt load plans due to the benefits they provide relative to the operational overhead they incur. Specifically, in this part of the thesis, we study the operational decisions that LTL carriers have to make dynamically on a day-to-day basis to route freight shipments from origins to destinations. Allowing for an additional next-terminal option for location-destination pairs means that carriers have to dynamically make decisions related to *which* of those two next terminals a particular shipment will be sent to – in addition to *when* it will be dispatched (the only decision for a 1-alt load plan). Moreover, these decisions have to be made recognizing that an uncertain number of new shipments will be collected from shippers and enter the line-haul network the next day. The problem is further complicated by the scale of operations

of these carriers; for instance, an LTL carrier can operate a linehaul network of around 150 terminals while handling around 10,000 commodities (defined here as shipments with the same origins, destinations, collection time, and due time) each week ([41]).

We seek to model this dynamic freight routing problem to address the following question: "What kind of decision-making tool is needed to make good routing decisions in a 2-alt load plan setting in the presence of demand uncertainty?". For the purpose of this study, we will assume that the tactical plan is determined a priori and is fixed for the operating season. This means that we are given, for each day of the week, the number of trailers that operate between every pair of terminals in the linehaul network. For simplicity, we consider an idealized setting in which we assume that all trailer movements take a single day to complete. This assumption is without loss of generality as the ideas can easily be extended to handle multi-period travel arcs if travel times are expressed as multiples of a base time period. In addition, we assume that we are given a set of commodities along with distributional information for their daily quantities (measured in pallets). Furthermore, the trailer movement patterns and the commodity patterns repeat from week to week in the operating season. Although our focus here is on the LTL/freight industry, our model is general enough to adapt to any dynamic routing system, e.g., that of the express parcel industry. We will refer to this problem as the Dynamic Freight Routing Problem (DFRP).

This part of the thesis is comprised of Chapters 3-5 and makes the following contributions to the literature. We first introduce the DFRP and model this problem as a Markov Decision Process (MDP) in this chapter. Due to the scale of the problem, solving the MDP exactly is hopeless except for the smallest of instances, and so we introduce an offline-online Approximate Dynamic Programming (ADP) solution approach which helps overcome the infamous "curses of dimensionality". The ADP solution approach consists of an offline *learning* or *training* phase in which the values of different states are learned via simulation, followed by an online *execution* or *testing* phase (which corresponds to implementation in practice). Chapters 4 and 5 present three different value function approximation (VFA)

mechanisms for use in the ADP algorithm we use for solving the DFRP.

In Chapter 4, we use a lookup table as the VFA method to store the learned values of the states in this ADP algorithm. At each decision epoch in the ADP algorithm, we solve an integer program to determine the best decision to implement. Such a decision is selected based on the one-period contribution to the reward, and an estimate of the reward-to-go represented by the values of the states in the lookup table. Chapter 4 also contributes to the ADP literature by investigating and providing a framework for integrating lookup tables into integer programs (IPs) to solve the decision subproblem in the ADP algorithm. To the best of our knowledge, this has not been previously studied. The framework consists of: (1) a modeling approach for the integration of lookup table value function approximations into decision subproblem IPs to form extended decision subproblem IPs, (2) a solution approach, PDS-IP-Bounding, which decomposes the extended subproblem IPs into many smaller IPs and uses dynamic bounds to reduce the number of small IPs that have to be solved, and (3) an adaptation of the ϵ-greedy exploration-exploitation algorithm for the IP setting. Furthermore, the use of lookup tables (and indeed the integration of lookup tables with an IP) poses size issues that we overcome by aggregating the space of post-decision states (the state of the system right after our decision is implemented and before the uncertainty of the next decision epoch is revealed). We, therefore, compare different aggregation schemes for our problem context, and show that despite the original post-decision state being high-dimensional, by using relatively simple features to "summarize" them, high-quality policies can be produced.

Following that, in Chapter 5, we replace the lookup table with linear and neural net VFAs. In both approaches, we use basis functions or features of the post-decision states to inform the VFA. We compare the quality of policies generated by these approaches to each other as well as to the ADP lookup table approach of Chapter 4. This contributes to the literature in two ways: (1) we contribute to the increasingly growing literature studying the use of neural networks for prediction and decision making, and more specifically, for use as

a VFA architecture in ADP algorithms, and (2) we compare the performance of parametric (linear and neural network) and non-parametric (lookup tables) VFAs using the same set of basis functions. As mentioned in [56], there is little literature directly comparing the two approaches, and this work contributes to that body of literature by conducting a comparison for a particular class of problems, namely, the DFRP.

This chapter serves as an introduction to this part of the thesis, and the remainder of this chapter is organized as follows. In Section 3.2, we review relevant literature relating to the DFRP. In Section 3.3, we state and formally model the DFRP as an MDP. We conclude this chapter presenting the ADP algorithm for the DFRP which we will use in Chapters 4 and 5.

3.2 Literature Review

As mentioned previously, our work focuses on the linehaul operations of LTL carriers. That is, we focus on the inter-terminal operations as freight shipments travel between linehaul terminals on their way from origins to destinations. This is in contrast with city operations, which involve the pickup/delivery of shipments from/to individual customers in a relatively small geographical region served by each of the terminals. Since city operations can be modeled as standard vehicle routing problems, they have been studied extensively in the literature. With recent advances in technology, dynamic variants of these problems have also been studied. For a review of dynamic vehicle routing problems we refer the reader to [57] and [58].

When considering a carrier's linehaul operations, the decisions that have to be made can either be strategic, tactical, or operational. Strategic decisions primarily consist of determining the location of the linehaul terminals and how the terminals should be connected. Examples of these problems for LTL carriers can be seen in [59], [60], [61], and, for a more general reviews of hub/terminal location problems, [62] and [63].

Much of the literature on the linehaul operations of LTL carriers focuses on *tactical*

planning. Service Network Design (SND), for instance, is a class of tactical planning problems (not necessarily specific to LTL carriers) that is used ahead of each operating season to determine: (1) the number and types of trailers needed by the carrier, (2) their schedules and movements, and (3) a freight flow plan, which dictates how freight with a given origin and destination will travel through the linehaul network. Therefore, the design of the service network determines how the carrier will meet the demand of the upcoming operating season. We refer the reader to [1] and [2] for reviews of this class of problems.

In the freight/LTL context, the freight flow plan is often referred to as a load plan. A common load plan configuration that LTL carriers use in practice is to stipulate that each origin-destination pair will have a single freight flow path, and that, for each destination, all freight flow paths into that destination must form an in-tree. This implies that there is only a single next terminal option in freight flow paths for shipments that are at an intermediate terminal and headed to the same destination. To the best of our knowledge, [50] was the first work on designing load plans for LTL carriers. In addition to introducing the load planning problem, they present a mathematical formulation for the problem along with a local improvement heuristic. The work was subsequently extended in [51], [52] and [11]. [44] model the load planning problem on a time-space network to accurately represent consolidation timings, and present a heuristic that uses decomposition techniques combined with slope scaling to solve large-scale instances. [40] and [41] develop IP-based neighborhood search methods for solving large-scale instances of the load planning problem. Our study in this work is motivated by our findings in Chapter 2 in which we consider slightly relaxing the in-tree load plan requirements by allowing freight at an intermediate terminal that is destined for a particular destination to be dispatched on trailers heading to one of *two* possible next terminal options. We have shown empirically that these 2-alt load plans produce considerable cost savings (on the order of 6%) over in-tree load plans when operating in an environment in which demand is uncertain. Furthermore, these cost savings are comparable to the savings achieved by relaxing the load plan requirements completely.

In this part of the thesis, we, therefore, seek to gain insight into how such a load plan can be implemented in practice operationally. Because these 2-alt load plans allow for two options instead of the traditional one option in the in-tree load plans, decisions have to be made not only about *when* to dispatch a pallet from a terminal, but also *where* to send it to next.

In terms of the decision hierarchy, the Dynamic Freight Routing Problem for LTL carriers that we present and study in this part of the thesis is a linehaul *operational* problem. In contrast to strategic and tactical planning problems, there is relatively little literature on the daily linehaul operations of an LTL carrier. The importance of using dynamic models in transportation/logistics, in general, has been highlighted in works such as [64] and [65]. As most research in this area assumes traditional load plan structures, the majority of studies relate to deciding the dispatch/closing timings of trailers (in environments where trailer dispatches are not scheduled), some even considering only a single link for the purposes of their analysis. [66] develop a Markov Decision Process model to study accepting customer requests and trailer dispatching for a distribution problem between multiple origins and destinations. [67] develop a simulation system that integrates many practical operational decisions (e.g. work rules, load plan, trailer-closing policies, loading times) to study the impact of these factors on service levels. Using this simulation model, they show that making decisions about routing shipments dynamically can significantly improve service levels. Furthermore, they formulate a dynamic programming model to determine a trailer-closing policy dynamically depending on the state of the system. This work was further extended by [68], where the authors develop routing strategies that route shipments adaptively using local/terminal real-time information. [69] then extended this work by incorporating a correction term in the model that accounts for the interactions of shipments in the system. In, [70] and [71], the authors present algorithms for creating cost-effective detailed driver schedules for tactical plans where they take various operational issues and regulations into account. [72] present a decision-making framework for accepting or rejecting new requests based on the real-time status of the linehaul network. Every time a new request arrives,

a mixed-integer program is solved to determine what changes would need to be made to profitably accommodate the new request, and a decision is made based on some acceptance criteria to accept the request (and make the necessary operational changes to trailer schedules and the load plan) or reject it. More recently, [73] propose an IP lookahead approach to determine the load plan dynamically based on the state of the shipments (both the ones in the system as well as the ones forecast to enter) and drivers in the system. As this lookahead IP is too large to solve directly, they suggest a time-decomposition approach in which small consecutive intervals of the horizon are solved at a time, and the results of each linked together via simulation. [74] design a decision support system for daily load planning. Because of demand uncertainty, they consider the use of alternate paths for freight if there is not enough capacity on the preferred path of the load plan, and use fast heuristics to allow for near real-time load plan adjustments to improve on-time performance. They find that it is possible to improve on-time performance without resorting to additional capacity.

A subset of the work on the operational aspects of LTL applications involves the use of Approximate Dynamic Programming (ADP). ADP is a (heuristic) framework which is used to overcome the curses of dimensionality exhibited by Markov Decision Process models, and has also been applied to a variety of transportation problems including full-truckload carriers (e.g. [75]), driver scheduling (e.g. [76]), and dyanmic VRP problems (e.g. [77]) among many others. We refer the reader to [78] for a comprehensive reference on ADP techniques, and [75] for a reference on ADP in transportation problems. [79] present an ADP approach for solving dynamic stochastic integer multi-commodity flow problems, a class of problems to which the DFRP belongs to. The authors compare the use of linear, piece-wise linear, and hybrid value function approximations.

With regards to ADP approaches used in the LTL context, [80] present a model that is similar to the one we present in this chapter, but only for a single node/terminal in the network. The model decides not only which shipments to route to which next terminals, but also dynamically decides the dispatching of trailers. Furthermore, they present a solu-

tion strategy that is based on ADP techniques and approximate the value function (which represents the downstream consequences of their decisions) using a linear model which estimates the unit value of having a particular commodity at a particular location. In our work, not only do we consider the state of the entire system, but we also use "features" to estimate the value of states. Our goal is to gain insight into which features better capture the state of the system; these can then be used to design or enhance heuristics for this problem, or to provide key performance indicators that quantify the state of the system as a result of decisions made on a given day. Some of the work has also focused on city operations, such as [81] and later [82] who study the dispatching of freight received by a terminal to local customers in the region. They model the problem as an MDP and use an ADP framework (with linear value function approximations to estimate downstream costs). They propose a number of basis functions for the linear value function approximation, and evaluate their performance.

Finally, in this part of the thesis, we make contributions to the general ADP methodology. In ADP, the value function is approximated by one of many possible approaches; the most common two in the transportation domains are using linear functions of some features of the (post-decision) state variable or using lookup tables. [78] and [56] discuss the advantages and disadvantages of each approach. Generally speaking, lookup table approaches can produce accurate approximations (when given enough computational time) but, in a basic implementation, would only provide values for states actually visited in the ADP algorithm, i.e., the accuracy of a value of a state in the lookup table is proportional to how many times that state has been visited in the training part of the ADP approach. On the other hand, while linear (or piecewise-linear) models impose structure on the value function, they offer a functional form that, once learned, can readily estimate a value for any state in the model. These linear value function approximation (VFA) schemes can either take the form of basis functions or features of the problem that are linearly combined with tunable weights to approximate the value function or they can be piecewise linear

functions, e.g. piecewise linear concave functions that capture the diminishing marginal return of an incremental "resource" at a particular location. Examples of the former include [83] in which the authors use basis functions to estimate the value function for an intermodal long-haul round-trips freight transportation setting, and [82] in which a variety of basis functions are evaluated for a dispatching problem faced by urban consolidation centers. Examples of the latter include [80] and [79], as well as [84] in which an ADP algorithm is presented for a maritime inventory routing problem. On the other hand, lookup table approaches have been primarily applied to dynamic vehicle routing settings as in [77] and [85]. We also note that combinations of linear and lookup table approaches have been explored ([56]).

Traditionally, most ADP problems in which the subproblem is a MIP rely on linear functional forms to estimate the value function since they are easily incorporated into the MIP (see [79, 80, 84, 83, 82] for examples). In Chapter 4, we contribute to ADP methodology by presenting a framework for integrating lookup tables into a MIP framework. To the best of our knowledge, we are the first to study integrating lookup tables into a MIP framework.

While most transportation applications either rely on the use of lookup tables or linear VFAs, neural networks have recently been gaining more popularity as a VFA option in a variety of domains, e.g. they have been commonly used in reinforcement learning applications. We refer the reader to [86], [87] or [88] for discussions on the use of neural nets as VFA mechanisms in reinforcement learning. They are attractive because of their power and their ability to use nonlinear activation functions and hidden layers to capture higher level features of the states without having to explicitly define those features ([89]). Indeed, they work quite well in environments where it is difficult to define suitable features for the problem at hand ([90]). Neural nets can be easily used as VFAs in ADP algorithms and, like linear VFAs and lookup tables, can be trained in an iterative manner ([78]) making them amenable to algorithms such as Approximate Value Iteration.

ReLU neural networks, in particular, are commonly used in practice for a variety of applications. In transportation applications, we are not aware of many research works involving the use of ReLU neural networks for value function approximation. [91], for example, study the use of neural networks as a VFA mechanism to learn dispatch policies in a transportation network. For large discrete decision spaces, [90] use ReLU neural networks as VFA mechanisms and integrate the network into an IP subproblem as part of an ADP algorithm to schedule an urban bus fleet in a stochastic environment. Furthermore, [89] offer an exploration on the use of neural net VFAs integrated in an IP setting. The authors compare (using instances of the "nomadic trucker" problem) neural net VFAs with linear VFAs using the same basis functions and conclude that using neural net VFAs offer advantages over linear VFAs in terms of reduced manual effort for feature design, and better policies. Similarly, [92] develop a reinforcement learning framework for problems with combinatorial decision spaces. They use a ReLU neural net to approximate the value function and integrate the neural net into their mixed-integer programming decision epoch subproblems. They demonstrate their approach using instances of the Capacitated Vehicle Routing Problem (CVRP), and show that while their approach is not competitive with state-of-the-art OR approaches for the problem, it performs better than other heuristics and reinforcement learning algorithms.

In Chapter 5, we compare three forms of value function approximation architectures in the context of the DFRP, namely, lookup tables, linear model VFAs, and neural network VFAs. This contributes to the ADP literature that compares the various VFA methods. As mentioned above, [89] compare the use of linear model VFAs with neural net VFAs, and [56] provide a thorough review on comparisons between parametric VFAs (e.g., linear models), and non-parametric VFAs (e.g., lookup tables). Their main observations from reviewing the literature is that there are very few studies that directly compare these approaches against each other (e.g., [93]), and that parametric VFAs can be successful when the functional form of the value function is known.

3.3 The Dynamic Freight Routing Problem for LTL Carriers

In this section, we first define the DFRP and then present an MDP model for the DFRP. We assume that the operating season's tactical plan (the service network design) has been determined exogenously and given is available as input to the DFRP, i.e., we are given the weekly trailer movement patterns and a 2-alt load plan (up to two next terminal options for all location-destination pairs). Although our presentation assumes a 2-alt load plan structure, any type of load plan can be used.

3.3.1 Problem Statement

The Dynamic Freight Routing Problem for LTL carriers (DFRP-LTL) is an operational problem in which multiple freight shipments are dynamically routed through the carrier's linehaul network from origins to destinations. The goal is to make routing decisions for the freight so as to maximize the profits for the carrier.

In our setup, we assume that trailer movement patterns and demand patterns repeat weekly throughout the operating season (demand patterns repeat weekly, but are stochastic). Since the operating season is relatively long compared to the representative week that we model, we will assume an infinite horizon for this problem. We also discretize time into days, i.e., each time period represents one day of operations. This level of discretization is appropriate for this application considering that, each workday, LTL carriers typically collect freight during the day from shippers, and collected freight enters the linehaul network in the evening hours (around 7 P.M.) when drivers bring that freight to their respective end-of-line terminals. The collection of freight from shippers during the day is known as the *city operation* (a typical vehicle routing problem with the terminal as the depot) and is not modeled explicitly in this research. We, therefore, take the start of each *operational* day (around 7 P.M.) to represent a decision epoch in the model, where routing decisions need to be made for freight that is currently at a line-haul network terminal (and that is yet to be

delivered to its ultimate destination). Such freight can either be: (1) new freight that has been collected in the city operation and brought to that terminal, or (2) transiting freight that is passing through that terminal on its way to its ultimate destination. The number of pallets of a shipment represents its quantity.

Since many aspects of this problem repeat cyclically, we use a cycle length T to capture the length of the desired cycle (e.g., a week) and use the set $\mathcal{T} = \{0, 1, 2, \ldots T-1\}$ to represent the *days* in a cycle (we use cycle and week interchangeably). We represent the operations of the line-haul network of the carrier by a wrap-around time-space network $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ with cycle length T. Each node in this network, $n = (i, \tau),\ i \in \mathcal{L},\ \tau \in \mathcal{T}$ represents a terminal location i (in a set of terminal locations, $\mathcal{L}$) and a day τ (in the set $\mathcal{T}$). The set $\mathcal{A}$ consists of two sets of arcs: (1) a set of *movement* arcs, $\mathcal{M}$, whose elements are of the form $((i_1, \tau_1), (i_2, \tau_2)),\ i_1 \neq i_2$ and represent scheduled trailer dispatches from terminal i_1 to terminal i_2 leaving on day τ_1 and arriving on day τ_2, and (2) a set of *inventory* arcs, $\mathcal{I}$, whose elements are of the form $((i_1, \tau_1), (i_2, \tau_2)),\ i_1 = i_2$ and represent holding freight at terminal i_1 from day τ_1 to day τ_2. Note that in some cases, we will use the term *flat network* to refer to the linehaul network without the time dimension, i.e., the graph $\mathcal{G}^{flat} = (\mathcal{L}, \mathcal{A}^{flat})$ where $\mathcal{A}^{flat}$ represents how the terminals are connected.

For simplicity's sake, we assume that all movement arcs $m \in \mathcal{M}$ have a travel time of one day. As long as all travel times consist of multiples of some base time period, this is without loss of generality as multi-period arcs can be handled by adding dummy nodes to the set $\mathcal{L}$ and breaking down such arcs to accommodate this. This approach of handling multi-period travel arcs is the one we will assume to simplify the presentation. Alternatively, these arcs can be handled by augmenting the state vector to include expected arrival times of pallets traveling on such arcs as in [79]. In our setting, therefore, freight will always be located at the terminals as opposed to being in transit on a trailer at the start of a decision epoch. Note that arcs starting from the nodes of day $T-1$ will wrap around and have their end points on day $(T-1+1) \bmod T = 0$. Associated with each movement

arc $m = ((i_1, \tau_1), (i_2, \tau_2)) \in \mathcal{M}$ is a number $Q_{i_1 i_2 \tau_1}$ representing the capacity given by the tactical plan on the scheduled trailers traveling on that arc. Similar to the movement arcs, we assume that all inventory arcs are of the form $((i_1, \tau_1), (i_2, \tau_1 + 1)),\ i_1 = i_2$, i.e., inventory arcs have a duration of one day. Note that we assume that inventory arcs are uncapacitated, and therefore, holding a pallet at a terminal is always a feasible option.

As mentioned above, in our setting, demand patterns also repeat from week to week, but are stochastic. Specifically, we define a *commodity* as a group of pallets that share a common set of attributes: origin o, destination d, availability *day* $e \in \mathcal{T}$, and due *day* $l \in \mathcal{T}$. Associated with each commodity is a random variable $D_{o,d,e,l} \in \mathbb{Z}_{\geq 0}$ with finite support which represents the number of pallets that originate at origin o on day e and are promised to be delivered by day l at location d, with distributional information available for each. We also assume that the random variables $D_{o,d,e,l}$ and $D_{o',d',e',l'}$ are independent for $e \neq e'$ (this allows for the optimal policy to be Markovian and deterministic; [79]). Note that the origin and destination are terminals in the network as far as the linehaul network is concerned; but the actual pickup/delivery operation from/to the shipper/consignee is part of the city operation, and is exogenous to the problem. We also note that the due day is not considered a hard deadline, and that freight is allowed to be delivered late subject to penalties. However, freight that is in danger of being "too late" (we use a parameter to specify how many days past the due day is considered "too late") will be outsourced to a third-party carrier and will incur outsourcing penalties. The details of the penalties associated with pallets delivered late or outsourced are discussed in Section 3.3.2. To illustrate the commodity information, consider the commodity $(i_1, i_5, 2, 0)$. This is a commodity that becomes available at terminal i_1 on day 2, and whose delivery is promised by day 0 (of the following week) at terminal i_5. The random variable determining the quantity of this commodity is $D_{i_1,i_5,2,0}$, and in a given week, the value of this random variable is realized.

Finally, we note that pallets can leave the system in one of two ways: (1) they are delivered either on time (by the due day) or late, or (2) their delivery is outsourced by a

third party. We also stipulate that the *sojourn time* for each commodity (defined as the total time a commodity can possibly be in the system) is no more than T, and therefore, an "instance" of a commodity is delivered (either by the carrier or by a third party) before a new instance can potentially appear.

3.3.2 Markov Decision Process Model for the DFRP

In this section, we formally model the DFRP as a Markov Decision Process (MDP). To that end, we will define each of the components of the MDP in turn.

State

As described in Section 3.3.1, we discretize time into days and take the start of each operational day to represent a decision epoch. The state of the system at the beginning of decision epoch $t \in \mathbb{Z}_{\geq 0}$ is partly determined by the number of pallets with a given ultimate destination and due day, or (d, l) pair, that are present at each of the locations in the set $\mathcal{L}$ at the start of that decision epoch. Note that an implicit assumption here is that we do not differentiate commodities by their origins; such a differentiation may be necessary if, for example, commodities with the same (d, l) pair but different origins or availability times have different revenues or penalties. While we do not consider such differences for simplicity's sake, they can be accommodated by adjusting the definition of the state variable to differentiate individual commodities but this adjustment is at the expense of increasing the size of the state vector. We let R_{it}^{dl} be an "inventory vector" representing the number of pallets that are present at terminal $i \in \mathcal{L}$ at the beginning of decision epoch t and that should be delivered to terminal $d \in \mathcal{L}$ by day $l \in \mathcal{T}$.

In addition, we need to specify the trailer capacity available at a decision epoch in our state variable as that information is needed to determine the set of feasible decision vectors. To do this, it suffices to include the corresponding day of the cycle for that decision epoch in the description of the state vector as that determines the trailer movement patterns for

the day. Therefore, the complete state of this system at decision epoch t, is taken to be $S_t = ([t], R_t)$ where $R_t = (R_{it}^{dl})_{l \in \mathcal{T},\, i,d \in \mathcal{L},\, i \neq d}$ and $[t] := t \bmod T$. The inclusion of the day of the cycle in the state variable also allows the problem to be transformed from a non-stationary periodic MDP model to an "augmented" MDP that is stationary ([94] and [95]). Since the problem then becomes a stationary (discounted) infinite horizon Markov Decision Process, we can rely on MDP theory to conclude that there exists a deterministic stationary policy that is optimal (under some mild conditions that are satisfied here; [96]). Having a stationary policy that is optimal on this "augmented" MDP is equivalent to saying that the policy may be day-differentiated for the MDP model that does not include $[t]$ in the state vector, i.e., if two vectors R_{t_1} and R_{t_2} are such that $R_{t_1} = R_{t_2}$, but $[t_1] \neq [t_2]$, then the decisions chosen by the policy need not be the same.

Decisions

Our only decisions in this model are routing decisions. Let x_{ijt}^{dl} represent the number of pallets with ultimate destination d and due day l that are loaded onto trailers traveling along arc $((i,[t]),(j,[t+1])) \in \mathcal{A}$; this represents movement of freight if $i \neq j$ and represents holding freight at terminal i if $i = j$. Therefore, a decision vector at decision epoch t is denoted by $x_t = (x_{ijt}^{dl})_{l \in \mathcal{T},\, i,j,d \in \mathcal{L}}$, i.e., a routing decision for all pallets that are currently in the system. If the process is in state S_t at the beginning of decision epoch t, the set of feasible decisions is then given by

$$\begin{aligned}
\mathcal{X}(S_t) = \{x_t : & \sum_{j \in \mathcal{L}:((i,[t]),(j,[t+1])) \in \mathcal{A}} x_{ijt}^{dl} = R_{it}^{dl} \qquad \forall i,d \in \mathcal{L},\ l \in \mathcal{T}, \\
& \sum_{l \in \mathcal{T}} \sum_{d \in \mathcal{L}} x_{ijt}^{dl} \leq Q_{ij[t]} \qquad \forall ((i,[t]),(j,[t+1])) \in \mathcal{M}, \\
& x_{ijt}^{dl} \in \mathbb{Z}_{\geq 0} \qquad \forall l \in \mathcal{T}, i,j,d \in \mathcal{L}, \\
& x_t \in \mathcal{Y}\},
\end{aligned} \tag{3.1}$$

where $\mathcal{Y}$ defines the set of routing decisions that complies with the imposed 2-alt load plan structure.

Transitions

We describe how the system transitions from decision epoch t to decision epoch $(t+1)$ in two steps. As the transition of the $[t]$ part of the state variable is clear, we focus on the transitions of the R_t part of the state variable. First, the system transitions deterministically from R_t to a post-decision state $\bar{R}_t$ which is the state of the system after implementing feasible decision $x_t \in \mathcal{X}(S_t)$ but just before the uncertainty of the next decision epoch is revealed, i.e., before new freight enters the system. Next, a probabilistic transition from $\bar{R}_t$ to state R_{t+1} occurs.

An important part of the deterministic transition in our setting is the outsourcing of pallets that are expected to be "too late". These pallets are assumed to be delivered by a third party and are subject to an outsourcing penalty. A pallet can be late by up to p days before it is deemed "too late" with p being the parameter controlling this tolerance. We assume that pallets are outsourced as soon as it becomes clear that the pallet will arrive too late. Outsourcing is handled as part of a transition: at decision epoch t, say a pallet with destination d is at location i, and a decision is made to send the pallet from location i to location j, then if the shortest path in the time-space network $\mathcal{G}$ from j to d is such that the earliest possible arrival time for this pallet is p days after its due day, then the outsourcing penalty is assessed at time t and the pallet will be outsourced and removed from the system, i.e., it will not appear in decision epoch $t+1$.

Therefore, if the process is in state R_t, and feasible decision $x_t \in \mathcal{X}(S_t)$ is taken, then

the process transitions to the post-decision state $\bar{R}_t = (\bar{R}_{it}^{dl})$ given by

$$\bar{R}_{i,t}^{dl} = \begin{cases} \sum_{j \in \mathcal{L}:((j,[t]),(i,[t+1])) \in \mathcal{A}} x_{jit}^{dl} & \forall d \in \mathcal{L},\ i \in \mathcal{L},\ l \in \mathcal{T},\ i \neq d,\ \Delta^{out}\big(t+1, i, d, p\big) = 0, \\ 0, & \text{otherwise}, \end{cases} \tag{3.2}$$

where $\Delta^{out}\big(t+1, i, d, p\big)$ returns the value 1 if a pallet will be outsourced (arrives p or more days after the due day l) based on the shortest path in the time-space network $\mathcal{G}$ from i to d starting at time $t+1$, and 0 otherwise. Given the wrap-around nature of the network, Δ^{out} would first determine whether or not a given due day l is in the past or in the future by leveraging the sojourn time assumption stated at the end of Section 3.3.1. Then, based on where that day actually is relative to the current day, it determines whether or not the pallet will inevitably be outsourced. The post-decision update above captures how many pallets of each (d, l) pair are at each terminal just before the uncertainty of the next decision epoch is revealed. The update also reflects the removal of pallets that are delivered to their ultimate destinations and pallets that will be outsourced.

Then, the (probabilistic) transition from the post-decision state to the next state is given by

$$R_{i,t+1}^{dl} = \bar{R}_{i,t}^{dl} + D_{i,d,[t+1],l} \quad \forall d \in \mathcal{L},\ l \in \mathcal{T},\ i \neq d. \tag{3.3}$$

Rewards

When the process is in state S_t and feasible decision $x_t \in \mathcal{X}(S_t)$ is taken, then the one-period contribution/reward to the objective function is composed of three parts:

1. Revenue for delivering a pallet to its final destination:

$$C_t^1(S_t, x_t) = \beta \sum_{d,l,i,j} x_{ijt}^{dl} \mathbb{1}_{\{j=d\}},$$

where β is a per-pallet revenue. Note that this does not include outsourced deliveries.

2. Penalty paid for delivering freight late to its ultimate destination:

$$C_t^2(S_t, x_t) = -c^{late} \sum_{d,l,i,j} \Lambda\left(t, l, p\right) x_{ijt}^{dl} \mathbb{1}_{\left\{j=d \,\wedge\, \Delta^{late}\left(t,l,p\right)=1\right\}},$$

where c^{late} is a per pallet per day penalty for late delivery, $\Delta^{late}\big(t, l, p\big)$ returns the value 1 if a pallet will be late if delivered to its ultimate destination d by the *end* of decision epoch t (0 otherwise), and $\Lambda\left(t, l, p\right)$ returns the number of days a pallet will be late if $j = d$ and $\Delta^{late}\big(t, l, p\big) = 1$ (0 otherwise). Therefore, this penalty is proportional to the number of days a pallet is late as well as the quantity.

3. Penalty paid for commodities that are delivered by a third party:

$$C_t^3(S_t, x_t) = -c^{out} \sum_{d,l,i,j} \delta(j, d)\, x_{ijt}^{dl} \mathbb{1}_{\left\{j\neq d \,\wedge\, \Delta^{out}\left(t+1,j,d,p\right)=1\right\}},$$

where c^{out} is a per pallet outsourcing penalty, and $\delta(j, d)$ is the shortest path from terminal j to destination d in the *flat network* $\mathcal{G}^{flat}$, i.e. the outsourcing penalty is proportional to the remaining travel distance in the linehaul network for that pallet.

Thus, the total contribution to the objective function is given by $C_t(S_t, x_t) = C_t^1(S_t, x_t) + C_t^2(S_t, x_t) + C_t^3(S_t, x_t)$. Note that we take the cost parameters β, c^{late} and c^{out} to be the same for all commodities for convenience.

Policy

A policy π is a function that maps each state S_t to a feasible decision vector $X^\pi(S_t) \in \mathcal{X}(S_t)$, where $X^\pi(S_t)$ is the decision induced by the policy π for state S_t. Note that since the time-space network repeats in a cyclical fashion, we seek a deterministic stationary policy where the policy may be day-differentiated from the perspective of the R_t part of the state variable (recall that the day of the cycle is part of the state variable, i.e., $S_t = ([t], R_t)$),

and where the expected total discounted reward is maximized, i.e.,

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left\{ \sum_{t=0}^{\infty} \gamma^t C_t(S_t, X^{\pi}(S_t)) \middle| S_0 \right\},$$

where S_0 is an initial state and $0 < \gamma < 1$ is an appropriate discount factor.

The Bellman optimality equation for this problem can be written as

$$V^*(S_t) = \max_{x_t \in \mathcal{X}(S_t)} \left\{ C_t(S_t, x_t) + \gamma \mathbb{E}\left[V^*(S_{t+1}) \middle| S_t \right] \right\}, \qquad \forall S_t. \tag{3.4}$$

where $V(S_{t+1})$ represents the value of being in state S_{t+1} (given by the transition partly described in Equation (3.3)), and represents the expected total future reward from that state onwards. We choose an optimal decision at each decision epoch according to the decision rule

$$X^{\pi^*}(S_t) = \arg\max_{x_t \in \mathcal{X}(S_t)} \left\{ C_t(S_t, x_t) + \gamma \mathbb{E}\left[V^*(S_{t+1}) \middle| S_t \right] \right\}, \qquad \forall S_t. \tag{3.5}$$

That is, we choose a feasible decision that optimizes the sum of the *one-period reward* and the discounted *expected reward-to-go* for the states we might transition to in the next decision epoch.

3.4 Approximate Dynamic Programming Heuristic

The MDP presented in Section 3.3.2 can only be solved for small instances. Complex dynamic optimization problems such as the DFRP suffer from the well-known curse of dimensionality, rendering the exact solution of all but the smallest of instances beyond our reach. Therefore, we have to resort to heuristic approaches such as ADP.

We use a version of ADP called Approximate Value Iteration (AVI). This approach is essentially a combined simulation and learning approach that steps forward in time learning the values of different states as the algorithm progresses. Since we modeled the problem as an infinite horizon MDP, we will use a finite simulation horizon H to approximate the

infinite horizon, where H (possibly $> T$) is a long-enough simulation horizon, chosen such that any additional discounted rewards that are accumulated in the simulation beyond H are negligible.

Before describing the specifics of the AVI algorithm, we first note that using the definition of the post-decision state, we can re-write Equation (3.5) as

$$X^{\pi}(S_t) = \underset{x_t \in \mathcal{X}(S_t)}{\arg\max} \left\{ C_t(S_t, x_t) + \gamma \bar{V}(\bar{S}_t) \right\}. \tag{3.6}$$

where $\bar{V}(\cdot)$ is a value function defined on the post-decision states $\bar{S}_t$ ([78]). The AVI algorithm, therefore, seeks to learn/approximate this value function through a number of simulation iterations. In Chapter 4, we use a lookup table to store and update the approximated values for the post-decision states that we visit in our simulation, while in Chapter 5, we explore the use of two forms of parametric VFAs, namely, linear models and neural nets.

The idea of AVI is as follows. For each post-decision state, we have a value $\bar{V}^{\pi_{t,N}}(\bar{S}_t)$ that corresponds to the current approximation of the value of being in this post-decision state (where $\pi_{t,N}$ is the incumbent policy at time t and iteration N). This value will be updated as the algorithm progresses, and will either be stored in a lookup table (Chapter 4) or will have a parametric form (Chapters 5).

We first set a maximum number of simulation iterations, N^{max}. We start the procedure with an initial state S_0 and initial approximation values for all post-decision states, $\bar{V}^{\pi_{0,0}}(\bar{S}_t)$. During a simulation iteration, N, at any decision epoch $t < H$, we find ourselves in a state S_t^N. We then need to choose a decision vector and an associated post-decision vector to transition into the next state at time $t+1$. In the case of the lookup table VFA, how we go about finding such a vector depends on whether we are in an *exploration* step (which we are in with probability ϵ) or an *exploitation* step (probability $1 - \epsilon$). That is, we deal with the exploration-exploitation question using a standard ϵ-greedy algorithm which we

adopt due to its simplicity. In an exploitation step, we solve Equation (3.6) using the most recent value function approximations for the post-decision states. On the other hand, in an exploration step, we randomly select a decision vector and an associated post-decision vector. The purpose of exploration steps is to avoid getting stuck in local optima and to force the algorithm to explore more of the post-decision state space thereby visiting states that it might not otherwise visit, and consequently observe their values. The details of how we select decision vectors and post-decision vectors in an exploration step in the lookup table VFA implementation follow in Section 4.1.2. Regardless of the current step type, we obtain a decision vector x_t^N, and a corresponding post-decision state vector $\bar{S}_t^N$. Let

$$\hat{v}_t^N := C_t(S_t^N, x_t^N) + \gamma \bar{V}(\bar{S}_t^N). \tag{3.7}$$

We use this value to update the value of the *previous* post-decision state $\bar{V}^{\pi_{t,N}}(\bar{S}_{t-1}^N)$. The form of this update depends on the VFA used and is, therefore, outlined in the two subsequent chapters. From the post-decision state vector $\bar{S}_t^N$, we then simulate the arrival of new commodities and their quantities according to their distributions, and complete the probabilistic transition to a new state S_{t+1}^N using Equation (3.3). This process for simulation iteration N repeats until decision epoch $t = H$, after which it terminates, and a new iteration $N + 1$ is started. After iteration $N = N^{max}$ is completed, the AVI algorithm terminates.

Our ADP algorithm consists of two phases. The first phase is an offline *learning* phase where the value function approximations are learned and updated. Its steps correspond exactly to the AVI algorithm that we outlined above. We note that the exploration probability, ϵ, can either be fixed throughout the training phase or be made to decrease with the iteration counter, N; we adopt the former for simplicity. The second phase is an online *execution* phase where the policy induced by the VFA (at the end of the learning phase) is fixed, i.e., the VFA is no longer updated. In addition, this phase consists purely of exploitation steps,

i.e. it is the optimization of Equation (3.6) that determines the choice of action and post-decision state at every decision epoch. Other than that, the steps of the execution phase are identical to that of the learning phase. In our work, we use the execution phase as a *testing* phase to evaluate the policy induced by the VFAs by performing a number of AVI iterations with the now-fixed policy.

CHAPTER 4
AN ADP SOLUTION APPROACH WITH A LOOKUP TABLE VFA ARCHITECTURE FOR THE DFRP

In this chapter, we describe the details of the lookup table VFA variant of our ADP algorithm for the DFRP. First, in Section 4.1, we describe the components of the algorithm. This is followed by a computational study in Section 4.2 in which we compare the various aggregation schemes discussed in Section 4.1.3, and demonstrate the effectiveness of the PDS-IP-Bounding algorithm described in Section 4.1.4. Finally, we summarize our work and present our conclusions in Section 4.3.

4.1 Algorithmic Outline

4.1.1 AVI for Lookup Table VFAs

In Section 3.4, we outlined the AVI algorithm for our ADP solution approach. Here, we briefly describe how the value function is updated after every time period in the AVI algorithm.

As described in Section 3.4, in iteration N and time period t, after solving the decision epoch subproblem, and choosing a decision vector x_t^N and a corresponding post-decision state vector $\bar{S}_t^N$, we use the value of $\hat{v}_t^N$ computed in Equation (3.7) to update the currently stored value for the previous post-decision state in the lookup table. The update is performed using the following expression:

$$\bar{V}^{\pi_{t+1,N}}(\bar{S}_{t-1}^N) = (1 - \alpha_{n-1})\bar{V}^{\pi_{t,N}}(\bar{S}_{t-1}^N) + \alpha_{t,N}\hat{v}_t^N \tag{4.1}$$

where $\alpha_{t,N}$ is a parameter called the step size (its value in our implementation is actually post-decision-state-specific and depends on the number of times a post-decision state's

value has been updated).

4.1.2 Integration of Lookup Tables with IPs

Our AVI implementation relies on a lookup table to store the value function approximations. Furthermore, as mentioned above, we need to solve Equation (3.6) at every decision epoch to determine the choice of decision and post-decision state vectors. For the DFRP, this involves solving an IP that integrates the most recent approximation values in the AVI algorithm into the model. In this section, we demonstrate how we can integrate these values to construct the IP we need to solve at every decision epoch.

During a simulation run, N, and at decision epoch t, the AVI algorithm is in state S_t, and we need to solve the decision epoch subproblem (3.6) to determine a transition to the next decision epoch. Note that we suppress the iteration number, N, in the notation for the state variable to reduce notational clutter as it should be understood that all of this occurs in some simulation run. Recall that the feasible set of the decision epoch subproblem (given by (3.1)) consists of integer points. Therefore, even ignoring the reward-to-go term in Equation (3.6), we would still need to solve an IP to find a decision vector. To add the reward-to-go term, we extend the IP to embed the most recent lookup table values in the model.

The first step is determining which post-decision state vectors are *compatible* with the current state S_t. For the current state, a post-decision state vector, $\bar{S}_t$ is compatible if there exists a decision vector that can be used to transition from the current state to that post-decision state using Equation (3.2). Because enumerating a number of potential post-decision states and checking their compatibility is considerable work, we instead rely on the IP itself to determine which post-decision state vectors are compatible with the current state. To that end, we enumerate a superset of the set of feasible post-decision vectors and use this superset to define the extended IP. The enumeration of this superset is done by checking what possible values each entry in the post-decision vector can take, and then

taking the cross-product of all those possible values. This naturally results in a large number of vectors, most of which will be incompatible, but we mitigate this by using aggregation (Section 4.1.3) and a decomposition solution approach (Section 4.1.4). We denote this enumerated superset by $\mathcal{P}_t$.

To integrate the lookup table values into the IP, we introduce the following new binary decision variables:

$$z_p = \begin{cases} 1, & \text{if post-decision state vector } p \text{ is chosen,} \\ 0, & \text{otherwise,} \end{cases} \qquad \forall p \in \mathcal{P}_t.$$

Furthermore, as we check compatibility of post-decision state vectors within the IP, we add variables $\bar{R}^{dl}_{i,t}$ that determine the value of the components of the post-decision vector in the model, and then match the resulting vector to a vector $p \in \mathcal{P}_t$ (and set its corresponding z_p value to 1). For exploitation steps, we can then write the *extended decision epoch subproblem* (EDES) IP as follows:

$$\max_{x_t, z} \quad C_t(S_t, x_t) + \gamma \sum_{p \in \mathcal{P}_t} \bar{V}^{\pi_t}(p) z_p, \tag{4.2a}$$

$$\text{s.t.} \quad x_t \in \mathcal{X}(S_t), \tag{4.2b}$$

$$\sum_{p \in \mathcal{P}_t} z_p = 1, \tag{4.2c}$$

$$\bar{R}^{dl}_{i,t} = \sum_{j \in \mathcal{L}:((j,[t]),(i,[t+1])) \in \mathcal{A}} x^{dl}_{jit}, \qquad \forall d \in \mathcal{L},\ i \in \mathcal{L},\ l \in \mathcal{T},\ i \neq d,$$

$$\Delta^{out}\big(t+1, i, d, p\big) = 0, \tag{4.2d}$$

$$\bar{R}^{dl}_{i,t} = \sum_{p \in \mathcal{P}_t} p^{dl}_{i,t} z_p, \qquad \forall i \in \mathcal{L},\ d \in \mathcal{L},\ l \in \mathcal{T}, i \neq d, \tag{4.2e}$$

$$z_p \in \{0, 1\}, \qquad \forall p \in \mathcal{P}_t. \tag{4.2f}$$

In the above model, $\bar{V}^{\pi_{t,N}}(p)$, represents the most recent entry in the lookup table for

post-decision state vector p. The objective function in this model is just a re-formulated version of that of Equation (3.6). Since only one z_p variable can be equal to one, the two expressions are equivalent. Constraints (4.2b) ensure that the selected decision vector is feasible. Constraints (4.2d) and (4.2e) are the aforementioned compatibility constraints that check for compatibility between the current state, feasible actions, and potential post-decision state vectors. In addition, they ensure that the correct z_p variable is set to one, i.e., for the vector p whose components exactly match the post-decision state vector determined by the model. We denote by (x_t^*, z^*) the optimal solution of this extended IP, and it is this solution that is used to transition to the next decision epoch in the AVI algorithm.

We use a similar model in exploration steps in the learning phase, but the difference here is that we would like to encourage the algorithm to choose a state vector that is not necessarily optimal with respect to Equation (4.2a). To accomplish that, we replace the values $\bar{V}^{\pi_{t,N}}(p)$ in (4.2a) with randomly generated coefficients drawn from a uniform distribution $U \sim [a, b]$ where a and b are either pre-specified numbers or set as the minimum and maximum current values in the lookup table. Although simple, this resulted in considerable improvements in the values of the resulting policies as seen in Section 4.2.4.

It is clear, however, that the size of this IP can become an issue. Specifically, the number of z variables is proportional to the number of enumerated post-decision state vectors (which may indeed be quite large). In addition, the number of constraints (4.2d) and (4.2e) is proportional to the dimensions of the post-decision state variables. To help alleviate this issue, in Section 4.1.3, we consider aggregation schemes that will help reduce both the dimensionality of the post-decision state vectors and the number of enumerated post-decision vectors at each decision epoch, thereby reducing the size of this extended IP. Furthermore, in Section 4.1.4, we present a solution approach for this extended IP that takes advantage of its inherent structure and that can be more efficient than solving the extended IP directly.

4.1.3 Aggregation Approaches

A major challenge when using a lookup table as the form of the value function approximation is that the accuracy of its entries is a function of how many observations we have for the corresponding post-decision states. Therefore, the more times we visit a post-decision state in the AVI algorithm, the more accurate its value will be in the lookup table. However, the number of possible post-decision states grows rapidly with the number of terminals and the number of commodities. This is not only a problem for the accuracy of the values in the lookup table, but also for the size of the EDES IP. Aggregation, therefore, helps reduce the number of post-decision states and their dimensionality by grouping "similar" post-decision vectors together and simplifying their representation. In this section, we present a number of aggregation approaches for the DFRP, and we compare their performance later in Section 4.2.2.

An aggregation of the post-decision state space is a function $\mathcal{U} : \mathcal{P} \mapsto \mathcal{Q}$ where typically $\mathcal{Q}$ is of a much smaller dimension than $\mathcal{P}$. An aggregation scheme addresses the question: "Is it possible to come up with more compact representations of the post-decision state space such that different post-decision state vectors that are similar (with respect to a set of problem features) are mapped into the same compact representation and share the same lookup table entry?". Therefore, after aggregation, the lookup table entries will consist of entries for the *aggregated* post-decision state vectors. Ideally, an aggregation should have the following features (but there are inherent trade-offs here): (1) the space $\mathcal{Q}$ is low-dimensional, and (2) the aggregation captures or retains most of the problem's important features and such features should inform the approximated values in the lookup table. Furthermore, because of the structure of our EDES IP, we also would like our aggregation to be expressible in the context of an IP, i.e., we should be able to replace constraints (4.2d) with whatever new definition we obtain for an aggregation post-decision state so that the extended IP can internally perform this mapping.

The expected reward-to-go in Bellman's equation can be seen as consisting of two main

parts: (1) the expected reward that we earn in the future from commodities that are in the system at time t but have not yet generated a reward, and (2) the expected reward that we earn from commodities that will enter the system in future. In this work, we focus on the former when developing aggregation schemes since as rewards are discounted, it is likely that commodities that are currently in the system will become reward-generating earlier than commodities that are not yet in the system. Therefore, "prioritizing" these commodities when choosing a post-decision state is likely to be more beneficial.

To ensure that the aggregation captures the essence of a post-decision state for commodities in the system, there are a number of important post-decision state features that we should consider. These include the current location of pallets (and how that relates to their ultimate destination), their due days, their quantities, and the transportation capacity of the network (which the commodities will ultimately compete for). Below, we propose a number of aggregation schemes for the DFRP, which we later compare computationally. In all the aggregation schemes presented below, although we focus on aggregating the inventory vector component of the post-decision state vector, $\bar{R}_t$, we note that our lookup table approximations are day-differentiated, i.e. the lookup table entries are stored in the form $([t], \mathcal{U}(\bar{R}_t))$. Including the day of the cycle helps capture demand and capacity patterns, and preliminary experiments showed benefits when including the day of the cycle in the aggregated representation.

SLACKS: In this scheme, we have an aggregated post-decision state vector with dimension $T-1$ and elements $\bar{R}_{jt} = \sum_{S(i,t,d,l)=j} \bar{R}^{dl}_{it}$ for $j \in [-p, T-1-(p+1)]$. In other words, this vector captures and tallies the number of pallets in the post-decision state vector that have *slack* values equal to j. Specifically, if we let, $\bar{l}$ be the *decision epoch* in which delivery of a pallet is promised (for a commodity with due day l), then we can define the slack $S(i,t,d,l)$ as $\bar{l} - (t+1+\delta_{[t+1]}(i,d))$. This value can be negative if the earliest the pallet can arrive is beyond the promised delivery decision epoch. While it captures due days and incorporates information on how soon the pallets can arrive at their destinations, this aggre-

gation does not consider any capacity information as it considers each pallet individually and does not account for any interactions between pallets.

TOTAL SLACK + CONGESTION (LOOKAHEAD): In this scheme, we use a compact two-dimensional aggregated post-decision state vector. The first component of this aggregated vector represents the total slack in the system, i.e., the sum of the slack values of all the pallets in the post-decision state (where slack values are defined as above). The second component is a congestion measure introduced to capture capacity interactions between commodities that are at the same location in the post-decision state, and are headed to the same destination (since they have the same alts and will likely compete for capacity). Specifically, this component captures the total capacity deficit across all terminal-destination pairs in the post-decision state. Some pairs may have no outgoing capacity in the post-decision state (on day $[t+1]$), but may have considerable outgoing capacity on day $[t+2]$. To capture this, we "look ahead" to find the earliest positive outbound capacity and use that to compute the capacity deficit. That is, if we let κ_{id}^t denote the total outgoing capacity on decision epoch t for terminal-destination pair (i,d) according to the 2-alt load plan, then we define the outgoing capacity of terminal-destination pair (i,d) in the post-decision state as $\kappa_{id} = \kappa_{id}^{t^*}$ where $t^* = \min\{\ell | \ell \geq t+1 \wedge \kappa_{i,d}^{\ell} > 0\}$. Then, we can write the second component in the vector as $\sum_{i,d} \max\{\sum_l \bar{R}_{it}^{dl} - \kappa_{id}, 0\}$. Note that this "lookahead" effectively collapses the time dimension and, therefore, does not take waiting into account. In some cases, waiting can still allow the pallets to reach their destinations and thus might be favorable, but waiting can be detrimental if it causes pallets to be late or outsourced. Preliminary experiments showed that incorporating this lookahead is beneficial in our instance settings.

TOTAL SLACK + CONGESTION (PALLET-DAYS): In this scheme, we again use a compact two-dimensional aggregated post-decision state vector similar to Total Slack + Congestion (Lookahead). While the first component here is exactly the same as the one used in the Lookahead variant, we modify the congestion component to address the waiting issue

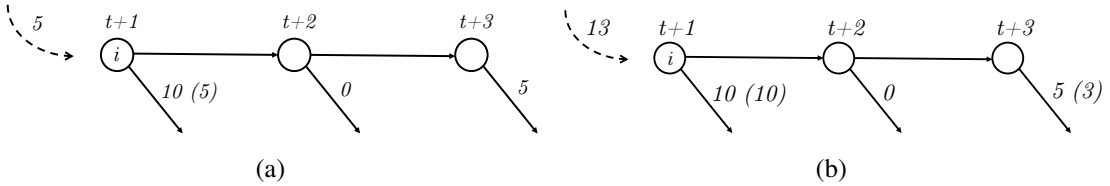


Figure 4.1: Illustration of pallet-days congestion measure

discussed above. Specifically, the congestion component now represents the total number of pallet-days required to clear out any deficits remaining from terminal-destinations pairs in the post-decision state. We illustrate this congestion component for a single terminal-destination pair in Figure 4.1. In this figure, a portion of the time-space network for terminal i is shown for 3 time periods starting from time period $t + 1$; the horizontal arrows indicate holding arcs while the diagonal arrows indicate the total outgoing capacity from terminal i that is available for destination d according to the 2-alt load plan. In the first example shown in Figure 4.1a, suppose a decision vector at time t sends a total of 5 pallets to terminal i (destined for d). Then, since there is sufficient outgoing capacity on that load plan to handle those 5 pallets, there is no deficit, and the congestion measure in this case is zero. However, in Figure 4.1b, we send a total of 13 pallets to terminal i at time t (destined for d), and we will have a deficit of 3 pallets in the post-decision state that need to be cleared from terminal i. Since there is no outgoing capacity at time $t + 2$, those 3 pallets are expected to remain in the system for two time periods until they can be moved out of terminal i, so the pallet-days congestion measure's value is $3 \cdot 2 = 6$. As this accounts for time and capacity, this is an improvement over the Lookahead congestion measure. However, the number of pallet-days vectors can be quite large, and in Section 4.2.2, we discuss how we use scaling/partitioning in our implementation to handle this issue.

Finally, observe that all three aggregation schemes are expressible in the EDES IP; we omit the presentation for the sake of brevity.

We have also experimented with a hierarchical decision-making process:

1. We first solve the EDES IP using the SLACKS aggregation scheme and obtain the optimal objective function value, $\hat{v}_t$.

2. We add the following constraint to the IP:

$$C_t(S_t, x_t) + \gamma \sum_{p \in \mathcal{P}_t} \bar{V}^{\pi_{t,N}}(p) z_p \geq \hat{v}_t,$$

   and re-optimize the IP with the objective of maximizing the *projected profit* for all commodities currently in the system. We define the projected profit as the profit obtained by emptying out the system, i.e., delivering or outsourcing all the pallets currently in the system at time t. To optimize for this projected profit, we need to optimize over a maximum horizon with length equal to the cycle length T (with all its associated arcs and their capacities) that ensures all commodities can leave the system either via delivery or outsourcing. This means new variables will be added to the model for future time periods (from t to $t + T$). We do not consider new arrivals in this optimization and are only concerned with commodities that are currently in the system.

However, our experiments showed that this hierarchical decision-making process did not yield policies that were consistently or noticeably better.

4.1.4 Extended IP Solution Algorithm

In this section, we present an *exact* solution algorithm for the EDES IP (4.2). This algorithm solves the extended IP by solving many small IPs where each IP corresponds to a single post-decision vector. Furthermore, it incorporates dynamic bounds to eliminate post-decision state vectors from consideration thereby reducing the number of small IPs that have to be solved.

The approach heavily relies on fixing a potential post-decision state vector, $\bar{R}_t$, in the EDES IP (4.2). Notice that by fixing a potential post-decision state vector, the problem

reduces to finding a decision vector (if the post-decision state vector is compatible with the current state) that optimizes the one-period reward $C_t(S_t, x_t)$. This IP is much smaller in size and we will refer to it as the post-decision state IP (PDS IP).

The approach is motivated, in part, by the fact that most likely only a small percentage of the commodities in the system at time t are reward-generating, and, therefore, that the majority of the reward comes from the reward-to-go part of the objective function (especially after the values in the lookup table have been updated). For this reason, we will sort the potential post-decision state vectors that we enumerate at time t in non-increasing order of their current lookup table values, i.e., $\bar{V}_t^{\pi_t}(p^{(1)}) \geq \bar{V}_t^{\pi_t}(p^{(2)}) \geq \cdots \geq \bar{V}_t^{\pi_t}(p^{(|\mathcal{P}_t|)})$. Our approach relies on going through this ordered list starting from $\bar{V}_t^{\pi_t}(p^{(1)})$, solving a small PDS IP for each vector to determine its value, and keeping track of the best objective found in the process.

Furthermore, we incorporate a simple bounding mechanism to enhance this search procedure. Specifically, we first solve the EDES myopically, i.e., with only the one-period reward term of the objective function and without any consideration of the reward-to-go of the post-decision state we will end up in. That will give us an upper bound on the one-period reward we can earn in that decision epoch, which we will call c^{UB}. Then, going down the sorted list of enumerated post-decision state vectors, as soon as a *compatible* vector is found with objective value $\hat{v}_t^p$ for the EDES IP, we can remove from the list all vectors whose lookup table value are less than or equal to $v_t^{cutoff} := \frac{\hat{v}_t^p - c^{UB}}{\gamma}$, as they cannot yield a better objective value for the EDES IP. This reduces the number of PDS IPs that we need to solve. For an illustration of this idea, see Figure 4.2. In this example, the first vector in the list was found to be incompatible. The second vector, however, was found to be compatible with an objective value of $\hat{v}_t^p$ for the EDES IP, and so any vector with a lookup table value less than or equal to v_t^{cutoff} was removed from the list.

This means that we can terminate the search procedure when all vectors have been processed or as soon as we find a vector that is compatible and has an objective function

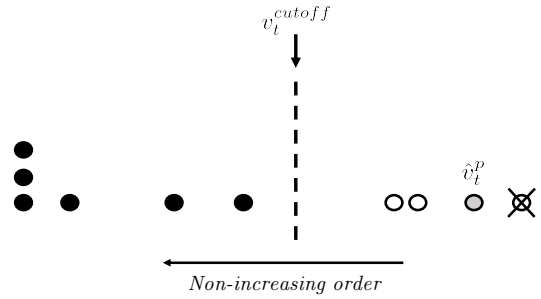


Figure 4.2: Example illustrating the bounding idea for post-decision state vectors whose PDS IPs need to be solved; circles represent the lookup table values of the vectors arranged on the number line (stacked circles indicate that multiple vectors have the same value); black-filled circles represent vectors that need not be considered

value for the EDES IP that is provably optimal. As we check post-decision state vectors in non-increasing order of their values, it is easy to see that when checking the j^{th} vector, an upper bound on the objective value of the EDES is given by $\gamma \bar{V}_t^{\pi_t}(p^{(j)}) + c^{UB}$. Therefore, this implies that as soon as a vector achieves a *PDS IP* objective value of c^{UB} the algorithm can terminate having found a provably optimal solution.

Putting all of this together, in Algorithm 1, we present the pseudocode detailing the steps of the algorithm (which we call PDS-IP-Bounding) to solve the EDES IP for decision epoch t.

Algorithm 1 PDS-IP-Bounding

Input: $\mathcal{P}_t$.

1: Initialize: $\hat{v}_t^{best} = -\infty$, $x_t^{best} = \emptyset$, $p_t^{best} = \emptyset$.

2: Determine c^{UB} by solving Model (4.2) myopically.

3: $PDSList \leftarrow$ Sort vectors in $\mathcal{P}_t$ in non-increasing order of their lookup table values.

4: **while** $PDSList \neq \emptyset$ **do**

5: $p \leftarrow$ First vector in $PDSList$.

6: Remove p from $PDSList$.

7: Solve PDS IP for vector p.

8: **if** p is compatible **then**

9: $\hat{v}_t^p \leftarrow$ Objective value for EDES IP for vector p.

10: **if** $\hat{v}_t^p = \gamma \bar{V}_t^{\pi_t}(p) + c^{UB}$ **then** ▷ Checking for optimality.

11: $\hat{v}_t^{best} \leftarrow \hat{v}_t^p$.

12: Update decision vector, x_t^{best}, and post-decision state vector, p_t^{best}.

13: Optimal solution found. Go to line 24.

14: **end if**

15: **if** $\hat{v}_t^p > \hat{v}_t^{best}$ **then**

16: $\hat{v}_t^{best} \leftarrow \hat{v}_t^p$.

17: Update decision vector, x_t^{best}, and post-decision state vector, p_t^{best}.

18: **end if**

19: Remove all post-decision state vectors with lookup table values less than or equal to $\frac{\hat{v}_t^p - c^{UB}}{\gamma}$ from $PDSList$.

20: **else**

21: Go to line 4.

22: **end if**

23: **end while**

24: **return** Optimal solution: $(\hat{v}_t^{best}, x_t^{best}, p_t^{best})$.

4.2 Computational Experiments

We conduct a set of computational experiments to:

1. demonstrate the effectiveness of the exploration scheme proposed in Section 4.1.2 for the IP decision subproblems,

2. compare (as part of the ADP solution approach) the various aggregation approaches proposed in Section 4.1.3 in terms of the quality of the policies they produce as well as the runtimes of their corresponding ADP algorithms, and

3. demonstrate the effectiveness of the PDS-IP-Bounding algorithm for solving the EDES IPs proposed in Section 4.1.4 compared to directly solving them as given by Equations (4.2) using a commercial solver.

We organize this section as follows. First, we describe the instance generation procedure and relevant problem parameters in Section 4.2.1. Next, in Section 4.2.2, we compare the aggregation approaches for the post-decision states, and benchmark them against a standard myopic policy. In Section 4.2.3, we analyze the performance of the proposed PDS-IP-Bounding algorithm for solving the EDES IPs. Finally, in Section 4.2.4, we demonstrate the effectiveness of the exploration scheme in the context of our EDES IPs.

All algorithms were coded in Python and all experiments were performed on a 20-core machine with Intel(R) Xeon(R) 2.30GHz processors and 256 GB of RAM running Red Hat Enterprise Linux Server 7.6, and with Gurobi 9.0.1 as the IP solver.

4.2.1 Instances and Model Parameters

After specifying a cycle length, T (we use $T = 5$), our instances are generated using a three-step procedure:

1. Generation of network topology: We generate the linehaul network, $\mathcal{G}^{flat}$, as a layered graph with four layers. The first layer contains a set of origin EOL terminals

(origin layer), the second and third layers contain sets of BB terminals, and the final layer contains a set of destination EOL terminals (destination layer). In our implementation, only nodes in consecutive networks are connected with the arcs always directed towards the destination layer, and we include all such arcs. An example of a layered network (for one of the configurations that we use) can be seen in Figure 4.3. We will use the configuration of the network, i.e. the number of nodes in each layer starting with the origin layer and ending with the destination layer, to partially differentiate our instances. For example, the network in Figure 4.3 will be referred to as a 4-3-3-3 instance. One could think about this layered structure as an abstraction of the transcontinental part of a U.S. carrier's service network; in particular, the origin layer may contain terminals on the East Coast while the destination layer may contain terminals on the West Coast, and the two intermediate layers are BB terminals that are somewhere in between.

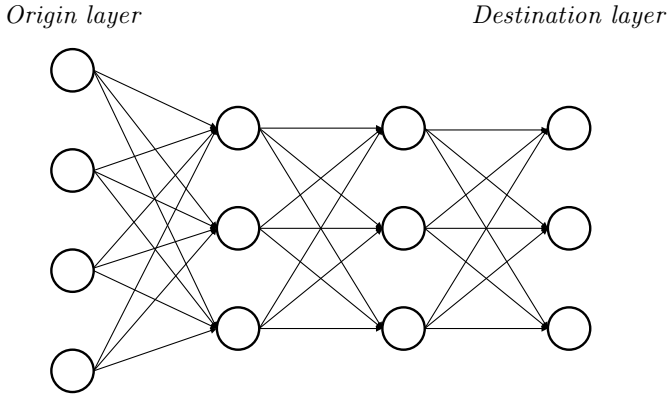


Figure 4.3: Example of layered network (4-3-3-3)

2. Generation of commodities: For each instance, and for each origin-destination pair, (o, d), we include a commodity of the form $(o, d, 0, 4)$ and one of the form $(o, d, 3, 2)$, i.e. since the minimum number of hops across the graph is three, each commodity has one day of slack while making that journey. Therefore, for a 4-3-3-3 instance, for example, the number of commodities is $4 \cdot 3 \cdot 2 = 24$. Depending on the instance, commodity demands either follow a uniform distribution over a common support of

$[0, q]$ where q is a user-defined parameter for the maximum number of pallets for a commodity, or a discrete triangular distribution ([97]) with a randomly generated mode, and a minimum and maximum of zero and q, respectively. We assume that all commodity random variables are mutually independent for convenience.

3. Determining the load plan and capacity: To determine the service network for the instances, our overall objective is to have, for each commodity, two types of paths: (1) tightly capacitated paths that enable pallets of that commodity to reach their destination one time period early, and (2) paths with more capacity that enable the commodities to arrive exactly on time (and some options that enable the pallets to arrive late). The network $\mathcal{G}^{flat}$ is first converted into a wrap-around time-space network, $\mathcal{G}$, by adding inventory arcs, and replicating arcs, $(i, j) \in \mathcal{G}^{flat}$ so that on each day $\tau \in \mathcal{T}$, there is a copy, $((i, \tau), (j, (\tau + 1) \bmod T))$, of the original arc (i, j). Then, we use the following two-phased approach to design the service network:

    a. We first remove the (intermediate-layer) BB terminals from the bottom half of the graph; when "splitting" the graph into halves and the number of terminals in a layer is odd, the bottom half has the smaller number of terminals. Next, we solve a 2-alt deterministic optimization problem as described in Chapter 2 using the expected demand values and a uniform trailer capacity of $\bar{Q}$ while stipulating the two following conditions: (1) nodes in the origin layer only have one alt available in this phase thereby choosing a single alt to the BB terminals in the top half of the first intermediate layer, and (2) 80% of the expected demand of each commodity arrives exactly on time, and 20% arrives one day late (we set $p = 1$).

    b. Add the BB terminals back, and augment the design obtained in Part a. by solving a 2-alt deterministic optimization problem using 20% of the expected demand values and with a trailer capacity of $\bar{Q}$ while requiring that: (1) flows

into the destinations arrive one day early, and (2) nodes in the origin layer have to have outgoing flows going through a BB in the bottom half of the first intermediate layer (the second alts for the origin nodes).

Thus, the top half of our network should have reasonable capacity to service commodity demands with the catch that they will "only" arrive on time, while the quicker paths in the bottom half arrive earlier but are tightly capacitated and may easily result in congestion. As alts primarily help alleviate congestion, this congestion element is useful in our instance design to see if our approach can exploit the alt structure and avoid situations which result in congestion.

We summarize the instance settings in Table 4.1. The first two digits of the instance label indicates the number of terminals in the origin layer, followed by three single digits for each of the remaining three layers; U/T represents the demand distribution (uniform and triangular, respectively); and this is followed by the value of $q = \bar{Q}$.

We use $\gamma = 0.8$ as the discount factor, $\beta = 3$ as the revenue for the delivery of a pallet (only applies if delivered by the carrier), $c^{late} = 2$ as the penalty for delivering a pallet late, and $c^{out} = 1$ as the penalty for outsourcing the delivery of a pallet. We set $\epsilon = 0.15$ as the probability that a training decision epoch is an exploration step; preliminary experiments with a few instances has shown this value to provide substantial improvements to the resulting policy. For AVI, we set $H = 20$ as the finite simulation horizon length, i.e., there are 4 cycles in each AVI iteration. We initialize all entries in the lookup table to be zero at the start of training, and set $\alpha := 1/\Theta(\bar{S}^a, N)$ where $\Theta(\bar{S}^a, N)$ is a function that keeps track of the number of times a particular entry $\bar{S}^a = ([t], \mathcal{U}(\bar{R}_t))$ in the (aggregated) lookup table has been observed up to AVI iteration N and time period t.

To ensure that our ADP algorithm learns a good policy no matter the starting state, we follow the approach of [82] to generate a set of realistic initial states, $\mathcal{S}^{initial}$, which we will sample from at the start of the ADP training and testing iterations. To accomplish this, we add an 'initialization' phase (prior to training) which is used to build the set $\mathcal{S}^{initial}$

Table 4.1: Instance settings

Instance	No. of EOLs	No. of BBs	Total No. of Commodities	q	Q
04333-U-5	7	6	24	5	5
06333-U-5	7	6	24	5	5
08433-U-5	11	7	48	5	5
10433-U-5	13	7	60	5	5
04333-U-10	7	6	24	10	10
04333-T-5	7	6	24	5	5
06333-T-5	7	6	24	5	5
08433-T-5	11	7	48	5	5
10433-T-5	13	7	60	5	5
04333-T-10	7	6	24	10	10
12544-U-5	16	9	96	5	5
06333-U-10	7	6	24	10	10
08433-U-10	11	7	48	10	10
10433-U-10	13	7	60	10	10
12544-U-10	16	9	96	10	10
12544-T-5	16	9	96	5	5
06333-T-10	7	6	24	10	10
08433-T-10	11	7	48	10	10
10433-T-10	13	7	60	10	10
12544-T-10	16	9	96	10	10

(initialized to be empty). We run 50 AVI initialization iterations; in each of these, AVI starts from a randomly generated initial state vector (which is most likely not realistic), and follows a myopic policy, i.e., a policy that completely disregards the reward-to-go component in the objective function of the decision epoch subproblems. The state encountered at decision epoch $t = \frac{H}{2} = 10$ in each iteration is then added to the set $\mathcal{S}^{initial}$ (this helps to avoid any warm-up or cool-down effects). Subsequently, in the training phase of ADP, we perform 1,000 AVI iterations (a total of 20,000 time periods), and at the start of each, we sample a random initial state from $\mathcal{S}^{initial}$ with a probability proportional to the frequency of initialization iterations with which it was observed. In the testing phase of ADP, we perform 200 AVI iterations (4,000 time periods) with the fixed policy, but each batch of 25 iterations will start from the same randomly sampled initial state. That is, for each of the initial states sampled in testing, we perform 25 sample paths with the goal of getting an accurate estimate for the value of that initial state. As a consequence, a maximum of eight

initial states will be drawn from $\mathcal{S}^{initial}$ in testing.

4.2.2 Analysis of Aggregation Approaches

To ensure a valid comparison in this experiment, for a given instance and phase (initialization, training, and testing), we use the same sample paths across all aggregation approaches. All ADP runs of this experiment utilized the PDS-IP-Bounding algorithm to solve the EDES IPs.

For the TOTAL SLACK + CONGESTION (PALLET-DAYS) scheme, we scale the aggregated post-decision state space along the congestion measure axis to reduce the number of PDS vectors. Specifically, we present two scaling variants:

1. Fixed Scaling: The total number of pallet-days is divided by 30 and rounded down, e.g., PDS vectors with the same total slack component and with a total number of pallet-days in the range from 0 to 29 all share a single lookup table cell.

2. Varying Scaling: The total number of pallet-days is divided by the number of terminal-destination pairs in the PDS vector for which we might find a positive number of pallets and rounded down, i.e., an "average" number of pallet-days is computed. The number of terminal-destination pairs in the PDS vector for which we might find a positive number of pallets can be determined at the start of each time period before setting up the PDS IP and passed along to the IP as a parameter.

In columns 3-6 of Table 4.2, we compare the average total testing (discounted) sample path reward (taken over the 200 testing sample paths) for the aggregation schemes: SLACKS, TOTAL SLACK + CONGESTION (LOOKAHEAD), TOTAL SLACK + CONGESTION (PALLET-DAYS VARYING), and TOTAL SLACK + CONGESTION (PALLET-DAYS FIXED). Because the SLACKS scheme was relatively time-consuming to solve on many of the instances, we limit our runs for this approach to instances for which the total runtime is less than 100 hours. We include averages for the first ten instances (which are those

for which SLACKS values were obtained) and overall averages for the other aggregation approaches in the penultimate pair of rows.

Table 4.2: Average ADP testing reward for aggregation approaches

Instance	Myopic	Slacks	TS+C (L)	TS+C (PD Var)	TS+C (PD Fix)
04333-U-5	59.93	84.47	96.68	96.77	98.62
06333-U-5	-3.57	64.65	86.44	79.67	74.45
08433-U-5	163.25	220.81	183.99	223.33	244.15
10433-U-5	73.85	200.16	174.53	206.16	197.45
04333-U-10	93.54	172.45	168.82	189.79	174.91
04333-T-5	64.66	103.62	107.92	108.76	108.91
06333-T-5	105.31	161.13	158.00	163.51	163.84
08433-T-5	261.09	228.31	224.02	217.06	228.40
10433-T-5	208.48	283.07	227.99	280.05	267.86
04333-T-10	127.79	176.38	179.43	174.66	178.52
12544-U-5	214.47	-	287.97	307.79	311.79
06333-U-10	-21.47	-	81.84	144.81	162.76
08433-U-10	351.83	-	334.35	429.82	408.97
10433-U-10	399.61	-	426.38	520.65	508.77
12544-U-10	502.76	-	337.89	539.35	648.14
12544-T-5	213.47	-	314.36	329.62	340.61
06333-T-10	16.17	-	148.65	173.54	178.36
08433-T-10	316.77	-	376.29	402.57	458.35
10433-T-10	489.81	-	406.19	589.12	585.99
12544-T-10	886.10	-	477.13	966.91	863.36
Average (First 10)	115.43	169.51	160.78	173.98	173.71
Overall Average	226.19	-	239.94	307.20	310.21
% Impr. Myopic (First 10)	-	46.84	39.29	50.72	50.49
% Impr.	-	-	6.08	35.81	37.14

Initially focusing on the first ten instances, we observe that the two TS+C (PALLET-DAYS) variants offer an improvement over the other two aggregation approaches. In particular, their policies offer an average improvement of about 8% over the LOOKAHEAD variant. On the other hand, the average improvement over the SLACKS aggregation scheme is less pronounced, although there is a noticeable improvement in the case of the U instances. The improvement in performance offered by the TS+C (PD) variants is even more pronounced when all 20 instances are considered. In particular, the policies found

by the two TS+C (PD) variants yielded a much higher overall average improvement of 28-29% over TS+C (L). These two approaches perform fairly evenly with TS+C (PD FIXED) performing slightly better overall.

Despite the TS+C (PD) variants performing only marginally better than SLACKS on the first ten instances, we argue that they are the better choices for an aggregation scheme. While the first component of the TS+C (PD) aggregation representation is an aggregated version of the SLACKS vectors, which may reduce its accuracy, the second component contains additional information about a different aspect of the system. In particular, TS+C (PD) captures capacity/congestion information whereas SLACKS does not. Furthermore, the TS+C variants offer a more compact representation of the post-decision state when compared to SLACKS. The competitive performance of SLACKS on the first ten instances is most likely because of the equivalence between capacity/congestion and slack values in the paths created by our instance generation procedure in the service network. Recall that we create paths that arrive a day early (a slack of 1) that are tightly capacitated, and paths that arrive on time (a slack of 0) that are less capacitated. Therefore, it is likely that the ADP algorithm was able to learn to avoid paths that offer lower slack values (and therefore more congestion) by virtue of this equivalence. However, in much larger real-world settings where there may exist multiple paths with the same slack values but different capacities, the addition of the congestion measure would provide more useful information about the state of the system.

We also include in Table 4.2 the reward values obtained by using a standard myopic policy which, at each time period, optimizes only the one-period reward without any explicit regard for the impact of the resulting decisions on future time periods. Due to the nature of the immediate reward of the DFRP, it is highly likely that there will be many optimal solutions for the myopic IP: for instance, there might be many feasible decision vectors that result in zero immediate reward (no pallets lost to outsourcing but no pallets delivered). To ensure that the myopic policy chooses decisions that are reasonable, we

introduce a small positive reward into the objective function for moving pallets closer to their destinations. This reward is proportional to the reduction in the shortest path distance to pallets' destinations in the service network $\mathcal{G}$ (and is allowed to be negative). It is only used in the myopic IP to choose reasonable decisions – decisions that move pallets closer to their destinations as is commonly done in practice by local terminal dispatchers – and is subtracted from the objective function value after the IP is solved. We also include in the last pair of rows the percentage improvement in the average reward values that the ADP approaches provide over the myopic approach.

We first note that all four ADP approaches yield improvements over the myopic policy. This demonstrates that the ADP approaches are capable (to varying degrees) of learning how to avoid some of the pitfalls that the myopic approach will run into. Even when the myopic policy was unable to find a policy with a positive total reward such as in the cases of 06333-U-5 and 06333-U-10, the ADP approaches were able to find policies that have high positive reward values. On average, the TS+C (L) approach yielded a 6% improvement over the myopic policy when all 20 instances are considered, but notably seemed to struggle on the largest two instances, 12544-U-10 and 12544-T-10 falling behind the myopic policy. On the other hand, the two TS+C (PD) approaches yielded substantial improvements of 35-37% over the myopic value. Furthermore, they are more consistent and perform much better on the largest two instances unlike their (L) counterpart.

Since the myopic policy prefers to keep pallets moving along the shortest path in the network $\mathcal{G}$, this will likely cause congestion as the pallets are funneled through the more capacitated paths in the service network. On the other hand, the ADP approaches are able to learn (through the features we use) the value of different post-decision states, identify better system states, and exploit the availability of the alts in the service network, thereby leading to higher reward values. Figure 4.4 shows the average performance metrics (taken over all 20 instances), % On Time, % Late, and % Outsourced, which highlight where the differences in the average reward values are coming from. These percentages are based on

commodities whose entire sojourn time fits within the time period window $[5, 15]$ within an AVI iteration to avoid warm-up and cool-down effects, i.e., they are computed for these commodities in each sample path in the testing phase, and the average over all testing sample paths is reported in the table. We observe that, on average, the two TS+C (PD) schemes exhibit a much higher on-time delivery percentage than the Myopic approach and even the TS+C (L) approach. Namely, the on-time percentage jumps from about 60% in the Myopic case to just under 77% in the TS+C (PD) approaches. This is coupled with a noticeable decrease in the percentage of outsourced pallets from 37% to about 21%. Thus, we see that, overall, the ADP approaches manage to deliver more pallets on time (and sometimes even more late, especially in the TS+C (L) and the TS+C (PD VAR) cases), while avoiding the capacity traps that cause pallets to be lost to outsourcing.

We depict a 3D visualization of the final (learned) lookup table for the instance 6333-T-10 with TS+C (PD FIX) in Figure 4.5. The figure shows that, generally speaking, states with the highest reward values (darker) are concentrated more in the area of the graph with high total slack and low congestion values which is to be expected. Therefore, choosing decision vectors that result in higher total system slack (e.g., by moving pallets closer to their destinations) and lower congestion measures (e.g., making use of alts to balance the distribution of pallets) are likely to yield higher rewards, and heuristics that attempt to find such decision vectors but that are more efficient and less complex than an ADP algorithm may possibly be developed along these lines.

Comparing the total runtimes (in hours) of these approaches in Figure 4.6, we immediately observe from the first figure that, for the first ten instances, SLACKS takes considerably more time to run while the other approaches have more comparable runtimes. On average, the three TS+C approaches offer almost a factor six speedup in runtime over SLACKS, and, at least in the case of TS+C (PD), similar (if not better) results. This gap in runtime is simply due to the SLACKS approach having a much larger superset size with many more PDS vectors enumerated compared to the other three approaches. For exam-

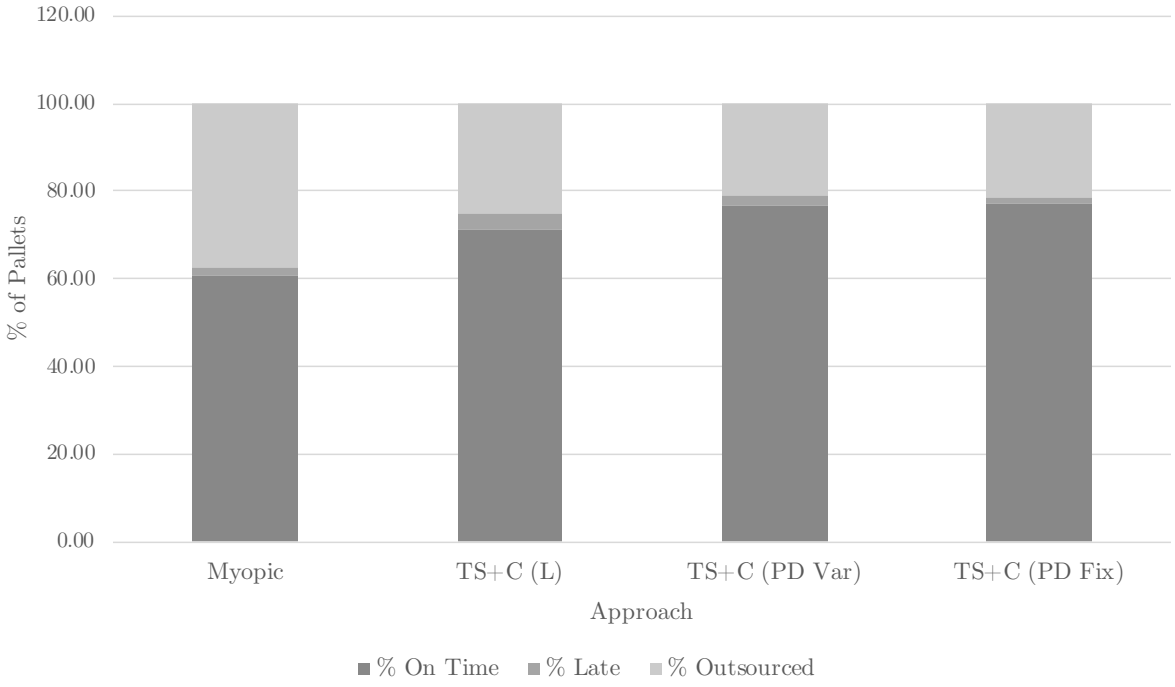


Figure 4.4: Performance metrics

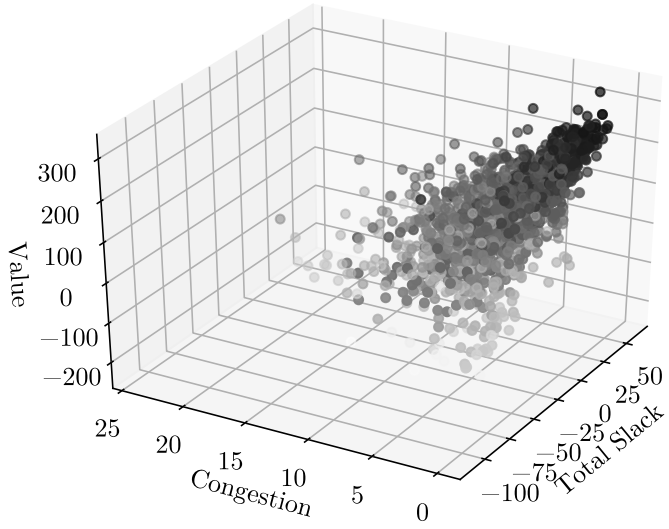


Figure 4.5: Visualization of final lookup table for instance 6333-T-10 with TS+C (PD Fix)

ple, in the case of the 10433-U-5 instance, the SLACKS approach enumerated, on average, supersets with about 261,751 PDS vectors per time period. This is in contrast with the 24,610 and the 4,856 vectors per time period enumerated by the TS+C (L) and TS+C (PD FIXED) approaches, respectively. From the second figure, we can see more clearly that the two TS+C (PD) variants consistently outperform the TS+C (L) in terms of runtime. Indeed, on average, the TS+C (PD) approaches are twice as fast as TS+C (L).

These experiments demonstrate that despite the original post-decision state vector being high-dimensional, a much simpler two-dimensional vector such as TS+C (PD) can produce a policy of good quality and a more efficient approach. Finally, we point out that these reward-to-go elements can also be used to augment (or design) other heuristics for this problem. For instance, in rolling horizon solution approaches in which only short horizons can be used at a time (because of the size of the instances to be solved), a reward-to-go approximation can help ensure that the decisions made in the short horizons are less myopic.

4.2.3 Effectiveness of PDS-IP-Bounding Solution Approach

In this section, we demonstrate the effectiveness of the proposed PDS-IP-Bounding solution approach presented in Section 4.1.4. Our ADP approach involves enumerating the vectors of a superset, $\mathcal{P}_t$, which may be very large, and consequently affects the size and solution times of the corresponding Extended IPs. The PDS-IP-Bounding solution approach provides a mechanism for working around this limitation by solving a number of small PDS IPs instead of a monolithic Extended IP.

For this comparison, at each time period, we solve the encountered decision epoch subproblem first as a monolithic Extended IP (using Gurobi), then we resolve the subproblem using the PDS-IP-Bounding algorithm and use this latter solution to transition to the next state. We note that all recorded times include any setup/overhead required by either of the two approaches, i.e., after enumerating the superset, we record the full time it takes either

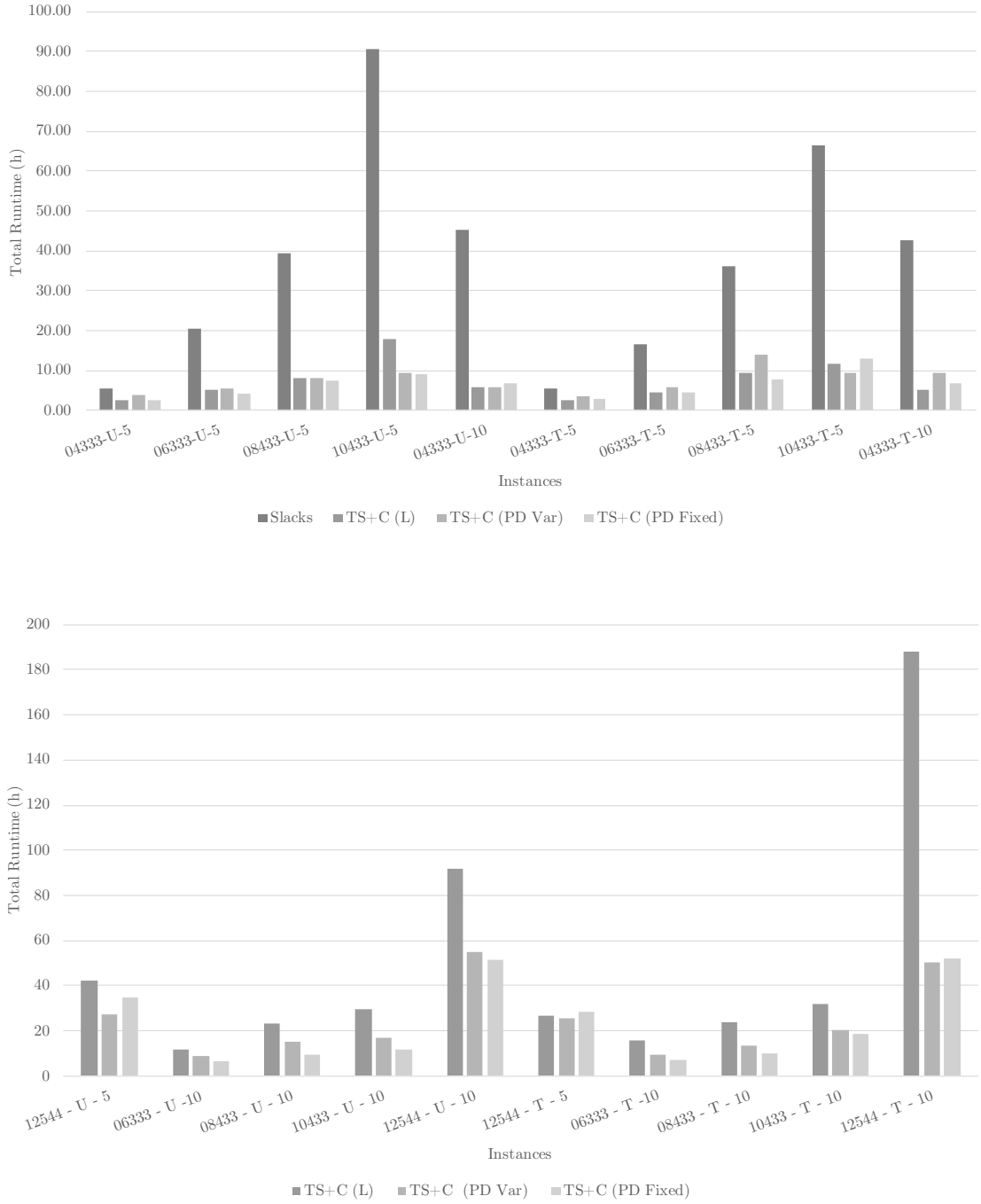


Figure 4.6: ADP algorithm runtimes for aggregation approaches

approach to choose a decision vector. In Tables 4.3-4.5, we report the following statistics (which are averaged over *all* decision epoch subproblems):

- **Ext. Time**: The average runtime (in seconds) of solving the decision epoch subproblems using the *Extended IP* approach.
- **PDS-IP Time**: The average runtime (in seconds) of solving the decision epoch subproblems using the *PDS-IP-Bounding* approach.
- **% PDSs Discarded**: The average percentage of PDS vectors in the superset $\mathcal{P}_t$ that were discarded during the solution process in the PDS-IP-Bounding approach (due to bounding).
- **# Enumerated Vectors**: The average number of enumerated PDS vectors, i.e. the average superset size.
- **Speedup Factor (SF)**: The speedup factor PDS-IP-Bounding provides over Extended IP, i.e. a speedup factor greater than one indicates that PDS-IP-Bounding is faster than Extended IP, and vice versa.

To keep running times reasonable, a time limit of 900s was imposed on the Extended IPs and a time limit of 120s on each individual PDS IP, as there were some cases in which the solver was unable to close the gap causing the branch-and-bound tree to exhaust the system's memory. We further note that because of the increase in runtime due to the inclusion of the Extended IP solution approach, a few more instances of the SLACKS aggregation scheme were excluded from this experiment. We also include separate averages for the six instances for which we present results (Avg. SF (Slacks)).

From Table 4.3, we observe that PDS-IP-Bounding provides a very significant speedup in runtime in the six SLACKS instances. Therefore, although the superset size is fairly large in the case of SLACKS (see Table 4.5), which consequently causes the Extended IP to be much larger in size, the PDS-IP-Bounding algorithm is able to sidestep the issue and we are

Table 4.3: Extended IP vs. PDS-IP-Bounding runtime comparison (overall)

Instance	Slacks			TS+C (L)		
	Ext. Time (s)	PDS-IP Time (s)	SF	Ext. Time (s)	PDS-IP Time (s)	SF
04333 - U - 5	1.30	0.50	2.59	0.19	0.17	1.11
06333 - U - 5	7.81	1.95	4.01	0.44	0.29	1.53
08433 - U - 5	15.56	2.89	5.38	0.62	0.43	1.44
10433 - U - 5	-	-	-	1.28	1.03	1.24
04333 - U - 10	-	-	-	0.65	0.31	2.09
04333 - T - 5	1.20	0.45	2.64	0.17	0.19	0.94
06333 - T - 5	9.16	1.14	8.03	0.35	0.29	1.21
08433 - T - 5	14.54	2.42	6.01	0.69	0.60	1.16
10433 - T - 5	-	-	-	0.97	0.55	1.78
04333 - T - 10	-	-	-	0.56	0.29	1.93
12544 - U - 5	-	-	-	2.59	2.51	1.03
06333 - U - 10	-	-	-	1.53	0.52	2.93
08433 - U - 10	-	-	-	2.66	1.01	2.64
10433 - U - 10	-	-	-	4.29	1.04	4.11
12544 - U - 10	-	-	-	10.61	5.78	1.83
12544 - T - 5	-	-	-	2.27	1.00	2.28
06333 - T - 10	-	-	-	1.70	0.78	2.19
08433 - T - 10	-	-	-	2.82	0.89	3.16
10433 - T - 10	-	-	-	3.87	1.29	2.99
12544 - T - 10	-	-	-	9.63	16.83	0.57
Avg. SF (Slacks)	-	-	4.78	-	-	1.23
Avg. SF	-	-	-	-	-	1.91

Instance	TS+C (PD Var)			TS+C (PD Fix)		
	Ext. Time (s)	PDS-IP Time (s)	SF	Ext. Time (s)	PDS-IP Time (s)	SF
04333 - U - 5	0.13	0.40	0.33	0.11	0.19	0.57
06333 - U - 5	0.23	0.47	0.48	0.19	0.31	0.63
08433 - U - 5	0.26	0.66	0.40	0.25	0.60	0.41
10433 - U - 5	0.28	0.86	0.32	0.30	0.80	0.38
04333 - U - 10	0.34	0.56	0.61	0.56	0.55	1.03
04333 - T - 5	0.11	0.33	0.35	0.09	0.23	0.41
06333 - T - 5	0.23	0.51	0.46	0.18	0.36	0.49
08433 - T - 5	0.29	0.84	0.34	0.43	1.02	0.42
10433 - T - 5	0.29	0.84	0.34	0.43	1.02	0.42
04333 - T - 10	0.35	1.10	0.32	0.24	0.78	0.31
12544 - U - 5	0.56	2.65	0.21	0.85	3.23	0.26
06333 - U - 10	0.66	0.67	0.99	0.51	0.47	1.09
08433 - U - 10	0.70	1.42	0.49	0.51	0.84	0.61
10433 - U - 10	0.92	1.55	0.59	0.80	0.95	0.84
12544 - U - 10	1.82	5.67	0.32	3.03	4.39	0.69
12544 - T - 5	0.70	1.92	0.37	0.88	2.13	0.41
06333 - T - 10	0.66	0.86	0.77	0.49	0.60	0.82
08433 - T - 10	0.69	1.20	0.57	0.63	0.80	0.79
10433 - T - 10	1.29	1.67	0.77	1.41	1.37	1.03
12544 - T - 10	1.85	4.77	0.39	2.66	5.07	0.52
Avg. SF (Slacks)	-	-	0.39	-	-	0.49
Avg. SF	-	-	0.47	-	-	0.61

Table 4.4: PDS-IP-Bounding percentage of discarded vectors (overall)

Instances	Slacks	TS+C (L)	TS+C (PD Var)	TS+C (PD Fix)
04333 - U - 5	97.47	97.79	97.68	97.60
06333 - U - 5	96.60	98.01	97.76	97.81
08433 - U - 5	97.68	98.86	98.54	98.30
10433 - U - 5	-	98.52	98.45	98.51
04333 - U - 10	-	98.45	98.06	98.05
04333 - T - 5	97.82	98.35	97.71	97.63
06333 - T - 5	98.32	98.12	98.30	98.26
08433 - T - 5	98.01	98.45	98.36	98.14
10433 - T - 5	-	98.58	98.97	99.33
04333 - T - 10	-	98.34	98.19	97.96
12544 - U - 5	-	98.50	99.41	99.50
06333 - U - 10	-	98.53	98.11	98.17
08433 - U - 10	-	98.74	98.52	98.43
10433 - U - 10	-	99.52	99.14	98.88
12544 - U - 10	-	98.09	98.67	97.97
12544 - T - 5	-	99.46	99.17	99.34
06333 - T - 10	-	98.17	98.15	97.99
08433 - T - 10	-	99.56	99.32	98.98
10433 - T - 10	-	99.15	98.97	98.76
12544 - T - 10	-	98.24	98.77	98.50
Avg. % PDS Discarded (Slacks)	97.65	98.27	98.06	97.96
Avg. % PDS Discarded	-	98.57	98.51	98.41

Table 4.5: Superset sizes

Instance	Slacks	TS+C (L)	TS+C (PD Var)	TS+C (PD Fix)
04333 - U - 5	18,298.08	3,832.79	1,650.30	1,089.20
06333 - U - 5	52,568.09	9,280.59	3,092.28	2,428.59
08433 - U - 5	145,641.56	14,292.49	3,104.20	2,717.38
10433 - U - 5	-	24,610.89	4,861.84	4,856.81
04333 - U - 10	-	15,376.80	5,716.83	10,184.19
04333 - T - 5	17,023.97	3,586.59	1,368.11	939.75
06333 - T - 5	69,291.93	8,549.13	2,615.89	1,749.49
08433 - T - 5	155,652.82	16,256.07	3,912.76	3,147.67
10433 - T - 5	-	4,216.98	4,475.94	23,972.14
04333 - T - 10	-	14,057.39	5,805.88	3,733.76
12544 - U - 5	-	56,529.30	7,440.34	10,744.64
06333 - U - 10	-	34,416.72	11,219.25	8,428.02
08433 - U - 10	-	60,253.35	11,446.49	10,226.44
10433 - U - 10	-	103,725.49	16,656.76	17,205.76
12544 - U - 10	-	208,908.03	28,215.89	42,759.37
12544 - T - 5	-	56,670.83	6,937.40	10,581.60
06333 - T - 10	-	40,167.59	12,880.83	9,608.72
08433 - T - 10	-	66,705.42	12,017.51	11,117.27
10433 - T - 10	-	101,300.79	21,694.69	22,916.36
12544 - T - 10	-	215,092.77	29,094.07	48,760.97
Avg. # Enum. Vectors (Slacks)	76,412.74	9,299.61	2,623.92	2,012.01
Avg. # Enum. Vectors	-	52,891.50	9,710.36	12,358.41

able to solve the subproblems more efficiently. We also see that PDS-IP-Bounding provides a decent speedup in runtime for the TS+C (L) approach in those same six instances. On the other hand, the Extended IP algorithm turns out to be faster in the two TS+C (PD) approaches. Similarly, for the averages over all 20 instances, we see that PDS-IP-Bounding provides close to a two-factor speedup in runtime when using TS+C (L), while becoming slightly more competitive in the case of TS+C (PD).

To further analyze these results, Table 4.4 shows the percentage of enumerated PDS vectors that were discarded during the solution process. On average, we observe that PDS-IP-Bounding only solves the PDS IPs corresponding to a very small percentage (less than 3%) of the superset PDS vectors with the vast majority being discarded using its bounding mechanism. This is true even in cases where Extended IP outperforms PDS-IP-Bounding in terms of runtime. While a majority of the time in the Extended IP approach (roughly 70-80%) is spent loading the IP due to the large superset size, the PDS-IP-Bounding approach is able to circumvent this issue and find provably optimal solutions much quicker in the cases of SLACKS and TS+C (L). As the % PDS Discarded values are very high, it may be possible to implement more intelligent enumeration schemes of the superset, thereby reducing these values. Regardless, the findings here suggest that the PDS-IP-Bounding approach is generally less sensitive to the size of the superset compared to solving an Extended IP. Clearly, the combination of efficient enumeration of vectors in the superset (irrespective of their feasibility), and using the PDS-IP-Bounding approach, provides an efficient solution approach, and helps sidestep the two issues of determining the feasibility of enumerated PDS vectors (by relying on the IP) and that of the size of the resulting IP (by relying on PDS-IP-Bounding).

Curiously, while the percentage of discarded PDS remains high in the two TS+C (PD) approaches, they do not exhibit the same speedup in runtime when we use PDS-IP-Bounding. However, further investigation suggested that this is a consequence of the scaling and not the congestion measure itself; for two of the smaller instances, we were able

to use a non-scaled version of TS+C (PD) and the results showed that PDS-IP-Bounding offers a speedup in runtime as is the case with SLACKS and TS+C (L) – although its reward value were notably worse than the scaled versions. There are two reasons why scaling/partitioning may have this impact. The first is that scaling reduces the number of PDS vectors which enables a much smaller Extended IP, thereby reducing its runtime. The second is that instead of fixing an individual PDS vector for each PDS IP, we now fix a scaled vector, e.g. the endpoint of the partition, and the search space for the PDS IP now involves finding a feasible decision vector whose resulting PDS vector maps to the scaled (fixed) vector. This may explain why we see more extreme outliers in terms of individual PDS-IP-Bounding runtimes. That is, due to the different nature of the PDS IPs, the solver either requires more time to find an optimal solution, or exhausts the time limit trying to close the optimality gap. However, removing these few extreme outliers, we can see a much better picture for PDS-IP-Bounding. For example, for the 12544-T-10 instance with TS+C (PD FIX), we show in Figure 4.7 box plots comparing the individual runtimes for each time period after removing outliers, and we observe that the quartile runtimes for PDS-IP-Bounding are noticeably lower than those of Extended IP (note that the crosses represent the mean runtimes). This suggests that PDS-IP-Bounding is still offering advantages despite the average time period runtime being distorted by the presence of these outliers. It also suggests that if sufficient computational power is available, running both approaches in parallel for each decision subproblem and choosing the solution found by the first approach to terminate might be a reasonable strategy for further efficiency gains.

There is also a noticeable difference between the overall average runtimes (which include training and testing) and their testing counterparts. In Tables 4.6 and 4.7, we show the runtime values and the additional statistics taken only over the time periods of the 200 testing iterations. A higher percentage of PDS IPs is discarded in the testing phase and, as a result, the speedup factors are noticeably higher. One reason for these results is the presence of exploration decision epoch subproblems in training. Recall that exploration

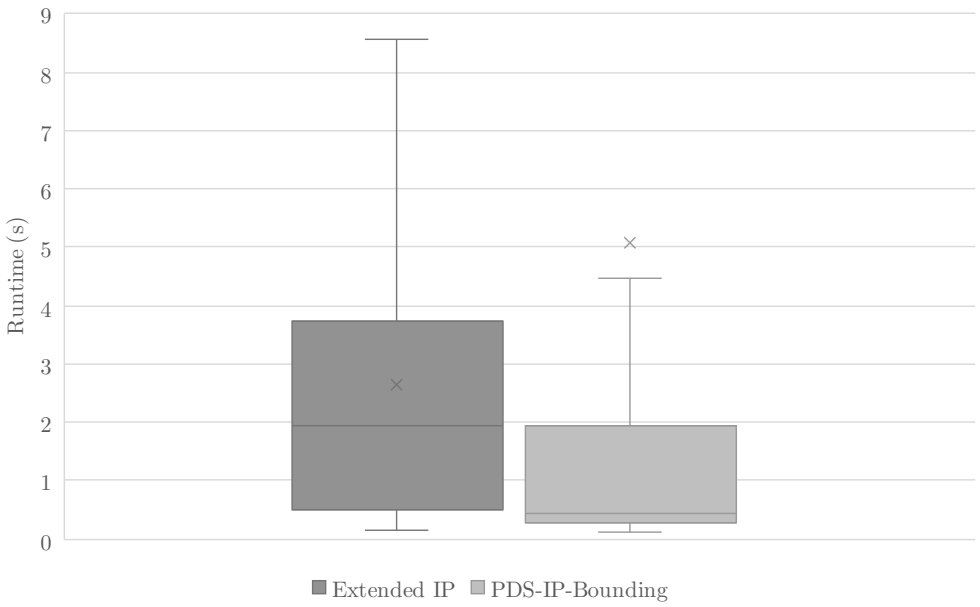


Figure 4.7: Box plot for individual time period runtimes for instance 12544-T-10 with TS+C (PD FIX)

decision epoch subproblems involve replacing the lookup table values with randomly generated coefficients in the range of values of the current lookup table (specifically, in the range of values of visited states, which are most likely nonzero values). This is in contrast with their exploitation counterparts which typically involve a small number of PDSs with a nonzero lookup table value. As a consequence of these artificial coefficients and the change in the number and distribution of nonzero PDS values, we observed a slowdown in the runtimes of exploration subproblems compared with their exploitation counterparts. This can be seen in Figure 4.8 where we show the average runtimes for exploration and exploitation subproblems for each of the two approaches. For brevity, we only show the results for the TS+C (PD FIX) aggregation approach.

A second reason is that the first few training iterations involve a lookup table in which very few states (if any) have been visited, and therefore, almost all enumerated PDS vectors in these subproblems will have a value of zero, making it harder for the algorithm's sorting and bounding mechanisms to be effective in those early iterations. An illustration of the effect of this warm-up phenomenon on the runtimes of the PDS-IP-Bounding approach can

Table 4.6: Extended IP vs. PDS-IP-Bounding runtime comparison (testing)

Instance	Slacks			TS+C (L)		
	Ext. Time (s)	PDS-IP Time (s)	SF	Ext. Time (s)	PDS-IP Time (s)	SF
04333 - U - 5	1.46	0.22	6.49	0.17	0.11	1.46
06333 - U - 5	10.79	0.41	26.62	0.47	0.22	2.12
08433 - U - 5	14.74	1.44	10.27	0.54	0.17	3.10
10433 - U - 5	-	-	-	1.34	0.33	4.08
04333 - U - 10	-	-	-	0.67	0.26	2.59
04333 - T - 5	1.29	0.18	7.10	0.17	0.14	1.20
06333 - T - 5	11.08	0.48	22.99	0.32	0.12	2.72
08433 - T - 5	12.65	0.54	23.26	0.68	0.36	1.88
10433 - T - 5	-	-		0.93	0.27	3.51
04333 - T - 10	-	-		0.52	0.16	3.25
12544 - U - 5	-	-		2.20	0.28	7.91
06333 - U - 10	-	-		1.38	0.16	8.44
08433 - U - 10	-	-		2.80	0.31	9.16
10433 - U - 10	-	-		6.05	0.49	12.42
12544 - U - 10	-	-		11.36	1.25	9.06
12544 - T - 5	-	-		2.23	0.43	5.24
06333 - T - 10	-	-		1.81	0.17	10.51
08433 - T - 10	-	-		2.47	0.17	14.16
10433 - T - 10	-	-		3.39	0.34	9.91
12544 - T - 10	-	-		11.54	0.79	14.68
Avg. SF (Slacks)	-	-	16.12	-	-	2.08
Avg. SF	-	-	-	-	-	6.37

Instance	TS+C (PD Var)			TS+C (PD Fix)		
	Ext. Time (s)	PDS-IP Time (s)	SF	Ext. Time (s)	PDS-IP Time (s)	SF
04333 - U - 5	0.13	0.51	0.25	0.10	0.11	0.93
06333 - U - 5	0.21	0.37	0.59	0.20	0.16	1.19
08433 - U - 5	0.29	0.44	0.66	0.28	0.58	0.48
10433 - U - 5	0.27	0.30	0.92	0.30	0.77	0.39
04333 - U - 10	0.34	0.35	0.97	0.45	0.27	1.63
04333 - T - 5	0.12	0.24	0.49	0.09	0.15	0.59
06333 - T - 5	0.24	0.28	0.84	0.18	0.22	0.82
08433 - T - 5	0.25	0.45	0.56	0.40	0.66	0.60
10433 - T - 5	0.25	0.45	0.56	0.40	0.66	0.60
04333 - T - 10	0.35	1.25	0.28	0.19	0.47	0.39
12544 - U - 5	0.52	4.61	0.11	0.83	0.77	1.08
06333 - U - 10	0.70	0.36	1.96	0.51	0.32	1.60
08433 - U - 10	0.83	0.96	0.87	0.52	0.52	1.00
10433 - U - 10	0.71	0.73	0.97	0.70	0.57	1.23
12544 - U - 10	1.65	5.78	0.29	2.11	3.39	0.62
12544 - T - 5	0.79	1.01	0.78	0.98	1.57	0.62
06333 - T - 10	0.85	0.83	1.02	0.46	0.46	0.99
08433 - T - 10	0.61	0.39	1.55	0.65	0.37	1.75
10433 - T - 10	1.28	0.97	1.33	1.25	0.57	2.19
12544 - T - 10	1.69	3.39	0.50	3.14	3.46	0.91
Avg. SF (Slacks)	-	-	0.57	-	-	0.77
Avg. SF	-	-	0.78	-	-	0.98

Table 4.7: PDS-IP-Bounding percentage of discarded vectors (testing)

Instances	Slacks	TS+C (L)	TS+C (PD Var)	TS+C (PD Fix)
04333 - U - 5	99.29	99.20	99.46	99.38
06333 - U - 5	99.49	99.68	99.65	99.57
08433 - U - 5	99.26	99.91	99.89	99.83
10433 - U - 5	-	99.95	99.84	99.86
04333 - U - 10	-	99.59	99.68	99.64
04333 - T - 5	99.63	99.54	99.44	99.35
06333 - T - 5	99.68	99.90	99.85	99.77
08433 - T - 5	99.55	99.91	99.84	99.72
10433 - T - 5	-	99.90	99.94	99.96
04333 - T - 10	-	99.62	99.82	99.79
12544 - U - 5	-	99.98	99.95	99.95
06333 - U - 10	-	99.86	99.89	99.85
08433 - U - 10	-	99.96	99.95	99.92
10433 - U - 10	-	99.98	99.95	99.95
12544 - U - 10	-	99.98	99.89	99.98
12544 - T - 5	-	99.98	99.94	99.94
06333 - T - 10	-	99.94	99.88	99.78
08433 - T - 10	-	99.98	99.97	99.97
10433 - T - 10	-	99.98	99.97	99.97
12544 - T - 10	-	99.99	99.99	99.97
Avg. % PDS Discarded (Slacks)	99.48	99.69	99.69	99.60
Avg. % PDS Discarded	-	99.84	99.84	99.81

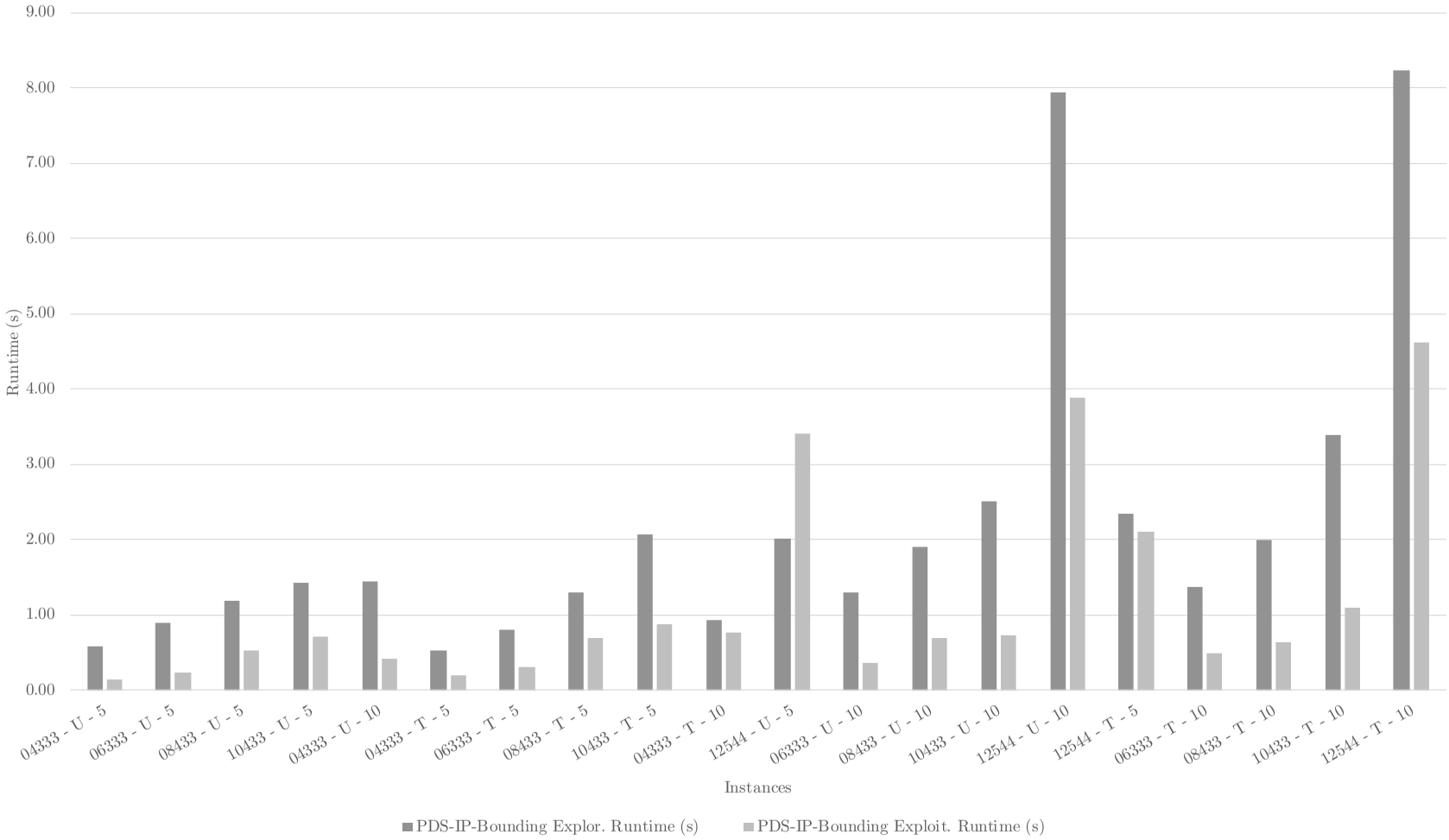


Figure 4.8: Comparison of exploration and exploitation subproblem runtimes

be seen in Figure 4.9, where we plot the average runtime of the first 20 AVI training iterations (i.e., each point is an average of the 20 subproblems comprising that AVI iteration) for the instance 12544-T-10 (with TS+C (PD FIX)). By the 14th training iteration, this

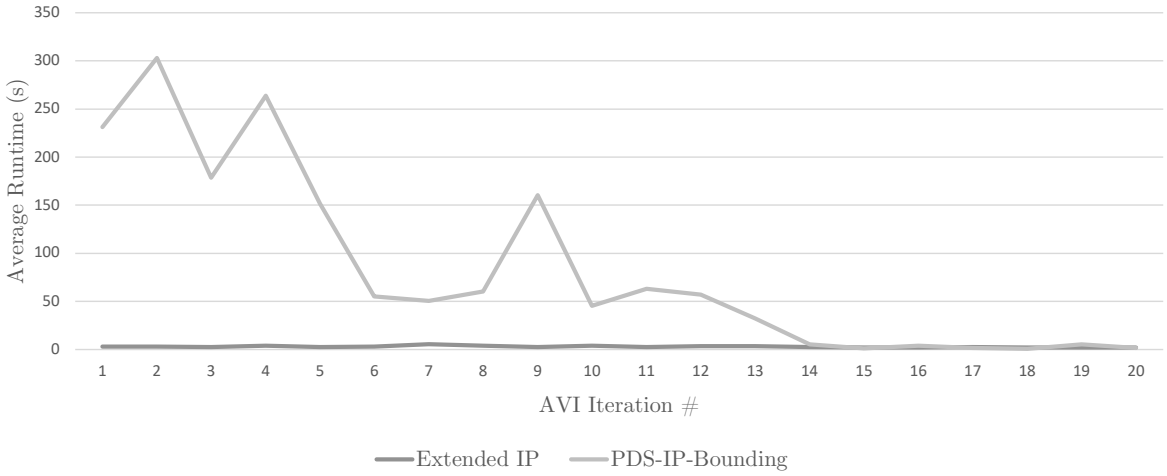


Figure 4.9: Training warm-up impact on runtime of PDS-IP-Bounding (12544-T-10 with TS+C (PD FIX))

effect is diminished and PDS-IP-Bounding performs on a similar level as Extended IP. It is possible to overcome this issue (and make the algorithm even more efficient) by checking if none of the enumerated post-decision states have been previously visited (and thus, have an initial value of zero). In such cases, the subproblem simply reduces to optimizing the one-period reward and the algorithm does not have to go through the list of enumerated PDS vectors as they become irrelevant, thereby drastically reducing the runtime in such iterations.

4.2.4 Effectiveness of Exploration Scheme

In this section, we demonstrate the value of including some exploration in the training phase of the ADP algorithm. While we adopt a standard ϵ-greedy algorithm, we adapt the standard mechanism of choosing a random decision vector to accommodate our IP setting as described in Section 4.1.2. In Table 4.8, we show that this mechanism with $\epsilon = 0.15$ yields benefits by providing a noticeable 3.5-fold improvement in the reward

values compared to having no exploration at all, with the highest improvements for the 12544-U-5 and 12544-T-5 instances. Once again, for brevity, we include only the results for the TS+C (PD) FIX approach.

Table 4.8: The value of exploration

Instance	Value w/ Exploration	Value w/o Exploration	Improvement Factor
04333 - U - 5	98.62	61.08	1.61
06333 - U - 5	74.45	47.67	1.56
08433 - U - 5	244.15	106.54	2.29
10433 - U - 5	197.45	61.37	3.22
04333 - U - 10	174.91	97.36	1.80
04333 - T - 5	108.91	66.24	1.64
06333 - T - 5	163.84	62.82	2.61
08433 - T - 5	228.40	100.82	2.27
10433 - T - 5	267.86	187.80	1.43
04333 - T - 10	178.52	90.67	1.97
12544 - U - 5	311.79	17.36	17.96
06333 - U - 10	162.76	32.74	4.97
08433 - U - 10	408.97	150.05	2.73
10433 - U - 10	508.77	258.90	1.97
12544 - U - 10	648.14	133.10	4.87
12544 - T - 5	340.61	50.46	6.75
06333 - T - 10	178.36	64.12	2.78
08433 - T - 10	458.35	217.45	2.11
10433 - T - 10	585.99	244.28	2.40
12544 - T - 10	863.36	280.16	3.08
Average	310.21	116.55	3.50

Naturally, exploration is vital in our instance settings because it enables the discovery of the less-capacitated (albeit longer paths) which consequently results in improved policies with higher rewards.

4.3 Summary and Conclusions

In this chapter, we introduced a lookup table to store VFAs for the ADP algorithm presented in Chapter 3 for the DFRP, and we compared a number of aggregation approaches and demonstrated that a two-dimensional aggregation scheme is able to produce policies

that are much better than a myopic policy. We have also introduced a framework for integrating the lookup table architecture into the IP decision epoch subproblems that need to be solved at every time period in the ADP algorithm. This framework includes: (1) the modeling of these integrated/extended IPs, (2) a solution approach which exploits their inherent structure and decomposes the problem into many small post-decision state (PDS) IPs, and (3) an adaptation of the standard ϵ-greedy algorithm to the IP setting. Our computational experiments show that there are computational advantages to be gained by prioritizing the solution of the most attractive PDS IPs and using dynamic bounds.

In the next chapter, we extend this work by using two parametric VFA mechanisms in place of the lookup table, and compare the performance of all three variants in the context of the DFRP.

CHAPTER 5
AN ADP SOLUTION APPROACH WITH PARAMETRIC VFA ARCHITECTURES FOR THE DFRP

In Chapter 4, we considered an ADP solution algorithm for the DFRP that utilizes a lookup table to store the value function approximations for post-decision states in an aggregated fashion. In this chapter, we study the use of two forms of parametric value function approximations for the DFRP, namely, linear and (ReLU) neural network value function approximations (VFAs). Both of these parametric architectures have the advantage of being easier to integrate into the decision epoch subproblem IPs defined in Chapter 4 compared to lookup tables, and, therefore, circumvent the limitation of size caused by enumerating PDS vectors to set up and solve the Extended IP. In turn, this allows larger instances to be tackled with more ease. In both approaches, we use basis functions as inputs to these VFA methods.

The remainder of this chapter is organized as follows. In Section 5.1, we present the two parmaetric VFAs, and describe how they are integrated into our ADP solution algorithm for the DFRP. In Section 5.2, we present a computational study in which we compare the runtimes and the quality of the policies produced by the two approaches both to each other and to the lookup table approach presented in Chapter 4. Finally, we present our conclusions and future research directions in Section 5.3.

5.1 Algorithmic Outline

In this section, we first describe the two parametric VFA architectures, and then discuss how the ADP framework presented in Section 3.4 is adapted for these new settings.

It is well-known that parametric and non-parametric VFAs each have their advantages and disadvantages, and consequently, each form lends itself to better performance on cer-

tain classes of problems ([56]). Non-parametric VFAs – typically lookup tables – directly estimate the value of usually-aggregated (post-decision) states and are theoretically capable of estimating highly nonlinear value functions as no functional form is imposed on the VFA. Practically, however, they are less reliable as the quality of the VFA is a function of the number of observations made per entry in the lookup table, and there are likely to be entries that have no observations at all over the course of an ADP run (or entries that have very few observations, and consequently, bad estimates). Therefore, while non-parametric VFAs offer the theoretical ability of accurately approximating any value function, the computational effort for doing so can be burdensome. On the other hand, parametric VFAs assume a particular functional form which can be a hindrance (depending on the relationship between the true value function and the parametric form of the VFA), but they allow the advantage of generalizing from a few observations, and therefore, are able to provide approximated values for every possible (post-decision) state.

5.1.1 VFA Architectures

As before, we will build our VFAs around the post-decision states, $\bar{V}(\bar{S}_t)$ (see Equation (3.6)). However, instead of using a lookup table to store the values of these approximations, we use one of two parametric forms, namely, linear model (L-VFA) and neural net (NN-VFA) architectures.

In both forms, we use explanatory variables or features, $f \in \mathcal{F}$, of the post-decision states to produce value function approximations for the post-decision states. In general, these features capture characteristics of the post-decision state and are then transformed into real numbers for use in the chosen functional form. Specifically, this transformation is done via basis functions, $\phi_f : \bar{S}_t \mapsto \mathbb{R}$, that map features of the post-decision state to a real number. A basis function for a feature can be the identity function, i.e. the basis function is equal to the feature itself (as is the case in our implementation), or it can be any other transformation of the features, e.g. an indicator function that is equal to one if there is more

than a certain number of pallets at a particular location in the post-decision state. Since our decision epoch subproblems are IPs, the only requirement we impose on such functions is to be expressible in an MIP setting. As in the case of lookup tables, using features is a form of aggregation that seeks to simplify the post-decision state space while retaining the important aspects of the problem that help inform the value function approximations. Therefore, the elements of the aggregation schemes presented in 4.1.3 (e.g., total slack and pallet-days congestion) are all examples of features that can be used for the DFRP.

Linear VFAs

In this case, the VFA assumes a linear form. Specifically, if for each basis function ϕ_f we define a corresponding weight parameter θ_f, then we can write the L-VFA as

$$\bar{V}(\bar{S}_t) = \theta_0 + \sum_{f \in \mathcal{F}} \theta_f \phi_f(\bar{S}_t), \tag{5.1}$$

where θ_0 is the y-axis intercept of the L-VFA.

Therefore, as long as the basis functions are linear in the decision variables of the IP, we can easily use this VFA form in the decision epoch subproblem IPs by replacing the reward-to-go term in the objective function (4.2a) with Equation (5.1). Note that we always use the most recent value for the weight parameters when setting up the IP decision epoch subproblem. Therefore, this IP for the case of L-VFAs will consist of the modified objective function along with constraints (4.2b), (4.2d), and the basis function constraints (which vary depending on the choice of features and basis functions).

Neural Net VFAs

In this case, the VFA takes the form of a Rectified Linear Unit (ReLU) feed-forward neural network. We refer the reader to [98] or [99] for general introductions to neural networks. The network consists of an input layer with $|\mathcal{F}| + 1$ neurons, i.e. number of features plus a

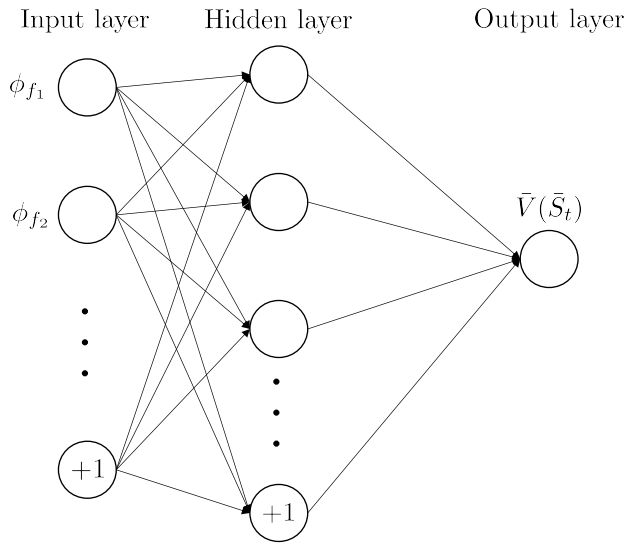


Figure 5.1: Example of a neural network with one hidden layer

bias scalar, $K \geq 1$ hidden layers each with a certain number of neurons, and an output layer with a single neuron that produces the VFA, $\bar{V}(\bar{S}_t)$. The network is also fully connected, i.e. each neuron in a layer connects to all the neurons in the preceding layer. Neurons in the hidden layers linearly combine values of neurons in the previous layer with tunable weights and apply a rectifier activation function which takes only the positive part of the resulting linear combination. This process repeats until the final layer takes the linear combination of inputs of the penultimate layer with the corresponding weights and produces the value function approximation $\bar{V}(\bar{S}_t)$. An example of such a network with a single hidden layer can be seen in Figure 5.1.

Next, we describe how the basis functions are transformed through the neural network and how it can be integrated into the decision epoch subproblem IPs. We use $\mathcal{K} = \{1, \ldots, K\}$ to denote the set of hidden layers in the neural network. Moreover, we define the input layer to be layer $k = 0$, and the output layer to be layer $k = K + 1$. We let Y_ρ^k denote the value of neuron ρ in layer k before its activation, i.e., before applying the rectifier function, and F_ρ^k denote the activated value of the same neuron. We also define P^k to be the set of neurons in layer k of the network. Note that the definitions of Y_ρ^k, F_ρ^k, and P^k

exclude the bias neurons. Further note that the activation functions are only applied in the hidden layers, and therefore, we have that $Y_\rho^0 = F_\rho^0 = \phi_\rho,\ \forall \rho \in \mathrm{P}^0$ (i,e., $|\mathrm{P}^0| = |\mathcal{F}|$). That is, the first layer values are equal to the basis function values which are fed to the neural network as input. Furthermore, we have that $Y^{K+1} = F^{K+1} = \bar{V}(\bar{S}_t)$, i.e. the output of the final layer is the value function approximation. Therefore, the neural net is characterized by the following equations:

$$Y_\rho^k = \sum_{\rho \in \mathrm{P}^{k-1}} W_{\rho,\rho'}^{k-1} F_\rho^{k-1} + \nu_{\rho'}^{k-1}, \qquad \forall \rho' \in \mathrm{P}^k, k \in \mathcal{K}, \tag{5.2}$$

$$F_\rho^k = \max\{0, Y_\rho^k\}, \qquad \forall \rho \in \mathrm{P}^k, k \in \mathcal{K}, \tag{5.3}$$

$$\bar{V}(\bar{S}_t) = \sum_{\rho \in \mathrm{P}^K} W_\rho^K F_\rho^K + \nu^K, \tag{5.4}$$

where $W_{\rho,\rho'}^{k-1}$ is the tunable weight between the neuron ρ in layer $k-1$ and the neuron ρ' in the subsequent layer k, and $\nu_{\rho'}^{k-1}$ represents the bias unit of neuron ρ' in layer $k-1$ (note that we write ν^K with no subscript as the output layer consists of a single neuron). Constraints (5.2) compute the linear combinations of neurons in the previous layer. Constraints (5.3) are the rectifier activation constraints which transform the values computed in Constraints (5.2) by taking their positive parts. Finally, Constraint (5.4) computes the value function approximation as a linear combination of the outputs of the final hidden layer. The weights $W_{\rho,\rho'}^{k-1}$ and $\nu_{\rho'}^{k-1}$ will be iteratively updated in the training part of the ADP algorithm as states are visited, their values recorded, and the neural net fit is updated.

If the basis functions of our features are expressible in the constraints of the decision epoch subproblem IPs, then as Constraints (5.3) can be easily linearized (modern solvers automate this process or see [100], [101], or [102] for examples on how this can be done), we can fully incorporate this neural net VFA mechanism into our decision epoch subproblem IPs. Specifically, the new objective function will consist of the right hand side of Equation (5.4) replacing the reward-to-go term in the objective function (4.2a), and the con-

straint set is composed of constraints (4.2b), (4.2d), (5.2), (5.3) after linearization, and the basis function constraints which are the inputs to the neural net, $Y_\rho^0 = F_\rho^0 = \phi_\rho,\ \forall \rho \in \mathrm{P}^0$. We note that more neurons and hidden layers generally increases the computational effort required to solve the resulting MIP ([89]).

5.1.2 Modifications to the ADP Algorithm of Section 3.4

To train the VFA architectures, we use the same ADP algorithm presented in Section 3.4 with minor modifications. The modifications mostly relate to the standardizing/scaling our input features and how the VFA is updated as training progresses. Furthermore, we do not have exploration steps in the implementation of the parametric VFAs.

With regards to standardization of input features, it is well-known that the scaling of input features can have practical implications on the numerical performance of neural networks, in particular ([99] and [103]). A common scaling practice is to standardize the input features, i.e. each feature is transformed to have zero mean and unit variance. We introduce a *scaling* phase before the training phase in the ADP algorithm as a way of obtaining estimates for the means and variances of the features that we use. This phase consists of a number of ADP iterations in which some policy is followed and observations about the values of being in the visited post-decision states are collected. For instance, if we follow a myopic policy in this phase, then we can collect observations of the values of the post-decision states that we visit, $(\bar{S}_{t-1}^N, \hat{v}_t^N)$ where $\hat{v}_t^N$ is defined as the total discounted reward from time period t until the end of the horizon in iteration N. Then, using the basis functions, we can transform these observations into a collection of data points of the form $\left(\left(\phi_f(\bar{S}_{t-1}^N)\right)_{f\in\mathcal{F}}, \hat{v}_t^N\right)$. At the end of the scaling phase, we can use these data points to determine the scaling parameters, i.e., the mean and variance, for each of our features and use these parameters throughout the training phase (particularly in the Incremental Approach explained below).

The adaptation of the ADP algorithm's training phase to update the weights of the pa-

rameteric VFAs is done using one of two approaches. The first of these is the classical Approximate Value Iteration approach in which the weights of the VFA are updated iteratively immediately after an observation is made. This approach mirrors exactly that of Section 3.4 with the parametric VFA being used as the approximation mechanism. That is, at a time period t in some iteration N in the training phase, we solve the decision epoch subproblem to choose a decision vector (and a post-decision state) using the current VFA predictions/weights. Then, we can compute the new observation, $\hat{v}_t^N$, using the objective function of the subproblem and form a new data point $\left(\left(\phi_f(\bar{S}_{t-1}^N)\right)_{f\in\mathcal{F}}, \hat{v}_t^N\right)$ that corresponds to the basis functions of the previous post-decision state and its value. This data point is then fed to the L-VFA or the NN-VFA, and the weights are updated using a stochastic gradient descent (SGD) algorithm with a constant learning rate η, before simulating the transition to the next time period in the simulation iteration. In the context of parameteric VFAs, we will refer to this approach as the Incremental Approach. Note that at the end of the training phase, the final set of VFA weights is fixed for the testing phase of the ADP algorithm where performance of the resulting policy is evaluated.

In the second approach, we will refit the VFA at the end of every simulation iteration N. Hence, rather than feeding the data points immediately to the neural net, they are collected throughout the simulation iteration. At the end of each simulation iteration, N, we use observations collected from iterations in the interval $(N-\xi, N]$ to fit the VFA from scratch (note that for the first few iterations of training, we use all available data until more than ξ iterations have been completed). That is, we use all observations in a sliding window of length ξ to refit the VFA and discard any observations prior to that. Fitting is also done using an SGD algorithm in this approach. When we refit the VFA in this approach, for each fit, we standardize features based on the means and variances of the observations within the window. At the end of each iteration, the VFA, and, consequently, the policy will be updated. We will refer to this approach as the Refitting Approach. As in the Incremental Approach, the final set of weights obtained at the end of iteration N^{max} is fixed for the

testing phase of the ADP algorithm.

5.2 Computational Experiments

In this section, using the DFRP, we compare L-VFA and NN-VFA to each other as well as to the lookup table approach described in Chapter 4. We first describe the details of our implementations of L-VFA and NN-VFA and the instances used in Section 5.2.1. Then, in Section 5.2.2, we present the comparison of the three approaches.

All algorithms were coded in Python and all experiments were performed on a 20-core machine with Intel(R) Xeon(R) 2.30GHz processors and 256 GB of RAM running Red Hat Enterprise Linux Server 7.6, and with Gurobi 9.0.1 as the IP solver.

5.2.1 Instances and Parameters

We use the same instances described in Table 4.1 for our experiments in this section. Where applicable, we use the same parameter settings that were described in that section. We also follow the same procedure for generating a set of initial states, $\mathcal{S}^{initial}$, and use the same number of iterations in the initialization, training, and testing phases.

In these experiments, we will use the TS+C (PD FIX) aggregation scheme for the lookup table approach, as well as for the basis functions in the two parametric approaches, i.e., we have day of the week, Total Slack, and Congestion (measured in pallet-days) as our three features for L-VFA and NN-VFA. Since the day of the week is a categorical variable, we use dummy encoding to allow the VFAs to handle this ([104]), and we leave out the column corresponding to the first day of the week. As we do not have a superset in the parametric case, we do not need to scale/partition the congestion measure as we do in the lookup table approach.

For the scaling phase, we follow the improved myopic policy described in Section 4.2.2. To compute the values of the visited post-decision states, we use the running reverse sums (until the end of the horizon), i.e. for the post-decision state visited at time t, the value

is the total (discounted) reward from time $t + 1$ to the end of the simulation horizon. We perform 10 scaling iterations in our parametric VFA runs.

For learning rates, we use $\eta^{LR,I} = 0.01$ for the L-VFA and the Incremental Approach, $\eta^{LR,R} = 0.01$ for the L-VFA and the Refitting Approach, $\eta^{NN,I} = 0.00001$ for the NN-VFA and the Incremental Approach, and $\eta^{NN,R} = 0.0001$ for the NN-VFA and the Refitting Approach. We also have two hidden layers for the neural networks, i.e., $K = 2$, and each hidden layer contains 10 neurons, i.e. $|\mathrm{P}^1| = |\mathrm{P}^2| = 10$. Both VFAs are initialized with all tunable weights set to zero at the start of the ADP training phase. Finally, we set $\xi = 50$ for the Refitting Approach. The parameters described here were all chosen based on preliminary tests with different choices, and the parameters were tuned independently for each approach.

5.2.2 Comparison of VFA Approaches

In Table 5.1, we present the average reward values for the two parametric VFA variants with the two fitting approaches, namely, L-VFA with the Incremental Approach (L-VFA-I), L-VFA with the Refitting Approach (L-VFA-R), NN-VFA with the Incremental Approach (NN-VFA-I), and NN-VFA with the Refitting Approach (NN-VFA-I). To make comparisons easier, the values in the table represent the percentage improvement in the average reward (taken over the same 200 testing sample paths) relative to the values of the lookup table approach with the TS+C (PD FIX) aggregation (LT-VFA) in Table 4.2. We do not report a result for instance 12544-T-10 for NN-VFA-R. We observed prohibitively long computation times for many of the EDES IPs in the testing phase of the ADP algorithm; the EDES IP time limit was reached in about 25% of the subproblems solved before abandoning the run. Upon closer inspection, we found that the optimality gaps for these "outliers" were very high upon termination, in the order of 50-60%. This suggests that a stronger formulation – or automatic reformulation – is necessary if the NN-VFA approach is to be viable for larger instances (research in this area is developing, e.g. [101, 102]).

Table 5.1: Average reward values for VFA mechanisms (relative to lookup table approach with the TS+C PD (FIX) aggregation)

Instance	L-VFA-I	L-VFA-R	NN-VFA-I	NN-VFA-R
04333-U-5	2.05	5.97	4.16	5.78
06333-U-5	31.69	26.94	31.21	25.97
08433-U-5	-18.86	-6.45	-13.77	-18.20
10433-U-5	-2.75	6.76	3.57	8.30
12544-U-5	3.36	1.53	0.13	-1.23
04333-T-5	-1.22	4.08	5.07	9.57
06333-T-5	-3.99	7.32	-6.17	-1.10
08433-T-5	-6.16	2.50	-18.14	-12.40
10433-T-5	0.86	1.61	0.09	3.88
12544-T-5	3.43	0.60	4.66	0.49
04333-U-10	11.28	11.21	14.05	12.08
06333-U-10	-12.35	-12.48	-6.34	-6.65
08433-U-10	-5.70	1.60	-6.19	-2.52
10433-U-10	3.57	6.62	11.22	4.81
12544-U-10	-15.74	8.39	-13.64	-2.54
04333-T-10	4.11	-1.66	-50.51	-7.13
06333-T-10	-9.35	7.71	1.38	12.16
08433-T-10	-9.09	-7.73	-6.03	-10.12
10433-T-10	1.44	1.42	0.39	-0.66
12544-T-10	9.57	21.37	16.05	-
Average*	-1.23	3.47	-2.36	1.08

* Averages do not include the 12544-T-10 instance.

We first observe that, on average, the Refitting Approach outperforms its Incremental counterpart on both VFAs. This is more observable in the case of the L-VFA. In fact, both the L-VFA-I, and the NN-VFA-I yield average reward values that are, on average, roughly on par, or slightly worse than the LT-VFA. In the case of the L-VFA, 13 out of the 20 instances resulted in an improvement in the average reward value when the Refitting Approach was used over the Incremental Approach, and of those seven instances in which it was outperformed, it is usually not far behind. On the other hand, there are instances where L-VFA-R approach convincingly outperforms L-VFA-I, e.g., instances 12544-U-10 and 06333-T-10.

We also see that L-VFA outperforms both NN-VFA and LT-VFA in terms of average reward values. In particular, the L-VFA-R approach offers a 3.47% improvement over the LT-VFA, on average (over 19 of the 20 instances), and improves on the value of the LT-VFA on all but four instances. Although excluded from the average computation, we point out that the 12544-T-10 – one of the two largest instances – exhibits a very high improvement in the order of 21% over the LT-VFA value. On the other hand, the NN-VFA approaches seem to be more or less comparable with the LT-VFA approach and also less consistent compared to L-VFA. In fact, the NN-VFA-R approach only manages to outperform the L-VFA-R approach on six instances.

There are two possible reasons why the L-VFA outperformed the NN-VFA (and indeed the LT-VFA) on these instances of the DFRP. The main reason relates to the nature of the basis functions that we use in these comparisons, namely total slack and congestion measured in pallet-days. It may be the case that these two features are very amenable to a linear model since, intuitively, more slack and less congestion in the post-decision state should be correlated with a higher reward value, and, therefore, fitting a simple linear model with those two features helps achieve policies of good quality. This can partly be observed in Figure 4.5, where post-decision states with higher values are those with higher total slack values and lower congestion values. In Figure 5.2a, we show the total slack values

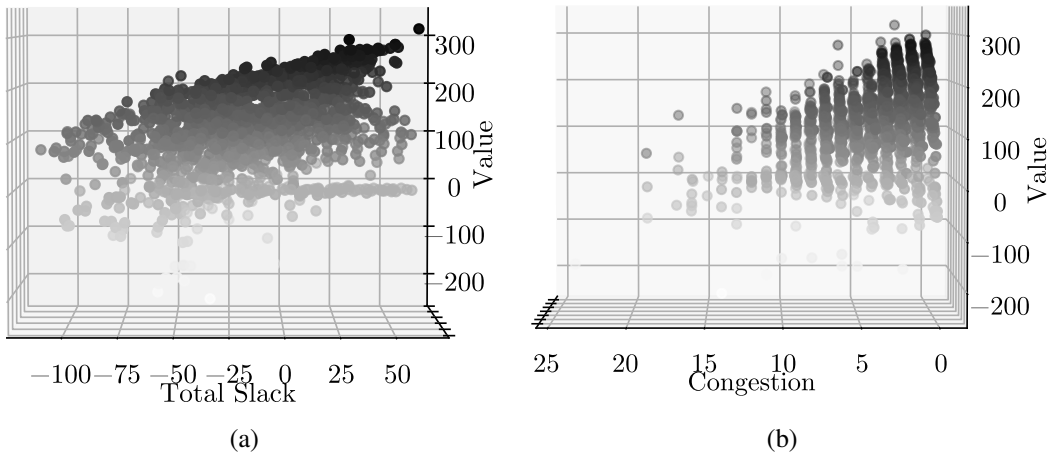


Figure 5.2: Visualization of axes of final lookup table for instance 6333-T-10 with TS+C (PD FIX)

(by rotating Figure 4.5) for this lookup table and observe that a linear fit might indeed do very well. The relationship between reward values and the congestion measure as shown in Figure 5.2b is a little less pronounced (which may be due to the scaling/partitioning of that axis), but we can still see why a linear model might do well there as well. As argued in [89], neural networks seem to perform well in settings where the features used are elementary descriptions of the post-decision state (the neural net can then extract higher-order features using its hidden layers thereby relieving the modeler from having to explicitly model these features), and therefore, it may be the case that the NN-VFA performs better if a different set of features is used – possibly features that are simpler and more varied in nature. The fact that for the DFRP we devised and used relatively complex, yet interpretable features that have a direct impact on the VFA in a simple linear relationship, may explain why the performance of L-VFA is superior to that of the NN-VFA.

Another reason why L-VFA performs better is that neural network performance can be sensitive to the choice of their hyper-parameters ([105]), e.g., the learning rate and the number of hidden layers and neurons. While we tuned these parameters, there might be sets of hyper-parameters that give rise to better performance but that we have missed in our tuning efforts. This provides another argument in favor of the use of L-VFA over NN-VFA

as it is more robust and simpler to implement, and does not have the added complexity of hyper-parameter optimization.

In Table 5.2, we compare the solution times for the EDES MIPs for the LT-VFA (once again with TS+C (PD FIX)), L-VFA-R anad NN-VFA-R. Note that the LT-VFA values are using the Extended IP formulation as it performed better in our experiments when compared to PDS-IP-Bounding for this aggregation scheme (Table 4.3). All solution times are in seconds.

Table 5.2: Comparison of EDES IP solution times (in seconds) for the VFA variants

Instance	LT-VFA	L-VFA-R	NN-VFA-R
04333-U-5	0.11	0.08	0.21
06333-U-5	0.19	0.12	0.56
08433-U-5	0.25	0.16	0.22
10433-U-5	0.30	0.20	0.56
12544-U-5	0.85	0.43	0.72
04333-T-5	0.09	0.08	0.22
06333-T-5	0.18	0.11	0.57
08433-T-5	0.43	0.16	0.29
10433-T-5	0.43	0.27	0.25
12544-T-5	0.88	0.36	0.62
04333-U-10	0.56	0.10	0.21
06333-U-10	0.51	0.13	0.57
08433-U-10	0.51	0.16	0.22
10433-U-10	0.80	0.27	0.65
12544-U-10	3.03	0.36	1.57
04333-T-10	0.24	0.09	0.26
06333-T-10	0.49	0.12	0.61
08433-T-10	0.63	0.17	0.23
10433-T-10	1.41	0.20	0.36
12544-T-10	2.66	0.36	-
Average*	0.63	0.19	0.47

* Averages do not include the 12544-T-10 instance.

We observe that both the L-VFA and NN-VFA approaches have faster average solution times for the EDES IPs compared to the LT-VFA. With the LT-VFA approach, the size of the EDES IPs grows with the size of the superset, which makes it less suitable for larger decision spaces. The L-VFA and NN-VFA approaches, on the other hand, are amenable to

large decision spaces as their parametric forms help mitigate the size of the subproblems, and, consequently, control their solution times. That solving the subproblems requires more effort for NN-VFA compared to L-VFA is natural and due to the increased size caused by the additional variables and constraints required to accommodate the neural network VFA. While computation times increase when the size of the instances increases, the increase is modest for the L-VFA and suggests that the L-VFA may be most appropriate when even larger instances have to solved. Therefore, we conclude that not only is the L-VFA the best performing VFA variant in terms of the quality of the resulting policy, but it is also the best performing VFA variant in terms of computational efficiency.

5.3 Summary, Conclusions, and Future Work

In this chapter, we presented two parametric VFA variants for the ADP algorithm for the DFRP presented in Section 3.4. Specifically, we present linear model (L-VFA) and neural network (NN-VFA) models that use basis functions to estimate the value of a post-decision state, and describe how these models can be integrated in the EDES IPs to produce VFAs for any post-decision state. We also compare these VFA variants to each other, as well as to one of the two best-performing lookup table variants (LT-VFA) from Chapter 4. Our computational experiments demonstrate that L-VFA outperforms both NN-VFA and LT-VFA both in terms of solution quality and the computational effort required.

There are a number of challenges to extending the approaches presented in this part of the thesis to solve real-world large-scale instances. It may, for example, require incorporating additional practical considerations (e.g., daily demand forecasts and detailed terminal operations) which may add other layers of decisions. They may also present new computational challenges, e.g., computing the congestion measure basis function might prove more challenging on larger instances as more variables and constraints are needed in the EDES IPs. In such cases, it may actually be beneficial to resort to less complex features and the performance of L-VFA and NN-VFA may be evaluated under such circumstances. An ex-

ample of such features may include the number of pallets at a location whose destinations are in a certain *direction* relative to their current location (where direction is defined using the alts or using geography). The logic here is that pallets that are headed in the same direction relative to their current terminals are likely to compete for capacity. Furthermore, we can classify the pallets into "urgent" and "less-urgent" pallets. Therefore, features may represent the number of urgent/less-urgent pallets at a location that are headed in a particular direction.

A natural future research direction is to investigate the effectiveness of rolling horizon techniques for the DFRP. Such approaches have the benefit of being easy to implement and understand, and perform well in a large number of application settings. In fact, our work in this part of the thesis is motivated by the use of a rolling horizon approach for large-scale instances of the DFRP. As LTL carriers may have a large network of terminals and handle a very large number of pallets per day, it may only be possible to solve rolling horizon subproblems for very short horizons (two to three time periods), and since these short horizons may result in relatively myopic solutions, it would be interesting to compare their performance against an ADP algorithm such as our L-VFA-R algorithm, and to study the benefits (if any) of combining both approaches, i.e. adding a VFA to the end of the horizon in the rolling horizon subproblem to compensate for these short horizons.

Appendices

APPENDIX A

THE VALUE OF LIMITED FLEXIBILITY IN SERVICE NETWORK DESIGN

A.1 Psuedocodes

A.1.1 SAA Algorithm for p-alt Problem

Algorithm 2 Sample Average Approximation Framework

Input: M, N, N'. ▷ Specifying algorithm parameters.

1: **for** $m = 1, \ldots, M$ **do**

2: Generate $\mathcal{S} \subset \Omega$, a random demand N-sample for sample problem m.

3: Generate (independently) demand sample of size $N' >> N$, for evaluation.

4: Solve sample problem m given by (2.3) to obtain a first-stage design for iteration m, $(\hat{r}^m, \hat{y}^m)$.

5: **for** $n = 1, \ldots, N'$ **do** ▷ Evaluation phase.

6: Solve second-stage subproblem (2.1e) given $(\hat{r}^m, \hat{y}^m)$ to compute $Q(\hat{r}^m, \hat{y}^m, \omega^n)$.

7: **end for**

8: Calculate $\hat{v}_{N'}(\hat{r}^m, \hat{y}^m) = \sum_{(i,j)\in\mathcal{A}} c_{ij}\hat{r}^m_{ij} + \frac{1}{N'}\sum_{n=1}^{N'} Q(\hat{r}^m, \hat{y}^m, \omega^n)$, the approximate expected total cost for design $(\hat{r}^m, \hat{y}^m)$.

9: **end for**

10: Select $(\hat{r}^*, \hat{y}^*) \in \underset{m\in\{1,\ldots,M\}}{\arg\min} \{\hat{v}_{N'}(\hat{r}^m, \hat{y}^m)\}$.

11: Compute the optimality gap estimate for $(\hat{r}^*, \hat{y}^*)$ and its variance using Equations (2.5) and (2.6).

12: **return** $(\hat{r}^*, \hat{y}^*)$, optimality gap estimate, and optimality gap variance estimate.

A.1.2 RandCut Separation Heuristic

Algorithm 3 RandCut

Input: $\mathcal{G}, \underline{r}, \underline{z}, RCMaxIter, \gamma^{RC}$. ▷ Specifying algorithm parameters.

for $\omega \in \mathcal{S}$ **do**
  $\mathcal{L} \leftarrow \emptyset$. ▷ $\mathcal{L}$ is the list of cutsets for which cut inequality is violated.
  for $t = 1, \ldots, RCMaxIter$ **do**
    $V \leftarrow \emptyset$.
    for $v \in \bar{V}$ **do**
      Choose a random byte $b \in \{0, 1\}$.
      if $b = 1$ **then**
        $V \leftarrow V \cup v$.
      end if
    end for
    if Equation (2.8) is violated for V **then**
      $\mathcal{L} \leftarrow \mathcal{L} \cup V$.
    end if
    if Equation (2.8) is violated for $\bar{V}$ **then**
      $\mathcal{L} \leftarrow \mathcal{L} \cup \bar{V}$.
    end if
  end for
  Sort cutsets in $\mathcal{L}$ in non-increasing order according to violation of Equation (2.8), and add the inequalities corresponding to the first γ^{RC} cutsets to the model.
end for

A.1.3 RGContraction Separation Heuristic

Algorithm 4 RGContraction

Input: $\mathcal{G}, \underline{r}, \underline{x}, \underline{z}, RGMaxIter, l, \gamma^{RG}$. ▷ Specifying algorithm parameters.

for $\omega \in \mathcal{S}$ **do**
  $\mathcal{L} \leftarrow \emptyset$. ▷ $\mathcal{L}$ is the list of cutsets for which cut inequality is violated.
  for $(i, j) \in \mathcal{G}$ **do**
    Compute slack for (i, j) with respect to constraint (2.1f) using $\underline{r}$, $\underline{x}$ and $\underline{z}$.
    Compute fractional parts for arcs given by $(\underline{r}_{ij} + \underline{z}^{\omega}_{ij}) - \lfloor(\underline{r}_{ij} + \underline{z}^{\omega}_{ij})\rfloor$.
  end for
  for $t = 1, \ldots, RGMaxIter$ **do**
    $ArcList \leftarrow$ Sort arcs by their slacks in non-increasing order, then by their fractional parts in non-increasing order.
    $\mathcal{G}' \leftarrow \mathcal{G}$. ▷ $\mathcal{G}' = (\mathcal{N}', \mathcal{A}')$ represents the contracted graph.
    while $ArcList \neq \emptyset$ **and** $|\mathcal{N}'| > \mathcal{N}^{final}$ **do**
      $RCL \leftarrow$ Select first l arcs in $ArcList$.
      $(u, v) \leftarrow$ Select an arc randomly from RCL.
      Contract arc (u, v) in $\mathcal{G}'$ while updating the slacks and fractional parts as described above.
      Update $ArcList$.
    end while
    $\mathcal{V}' \leftarrow$ Enumerate cutsets on $\mathcal{G}'$.
    for $V \in \mathcal{V}'$ **do**
      if Equation (2.8) is violated for V **then**
        $\mathcal{L} \leftarrow \mathcal{L} \cup V$.
      end if
    end for
  end for
  Sort cutsets in $\mathcal{L}$ in non-increasing order according to violation of Equation (2.8), and add the inequalities corresponding to the first γ^{RG} cutsets to the model.
end for

A.1.4 Heuristic-SS Algorithm

Algorithm 5 Heuristic-SS

1: $SSIterate \leftarrow True.$

2: Initialize multipliers: $\rho^0_{ij\omega} = \frac{\hat{c_{ij}}}{Q} \quad \forall (i,j) \in \mathcal{G},\ \omega \in \mathcal{S}.$

3: $t \leftarrow 0$ ▷ Initialize iteration counter.

4: **while** $SSIterate$ **do**

5: Solve (2.9) with multipliers $\rho^t_{ij\omega}$ to obtain a feasible integer set of first-stage variables for iteration t of slope scaling, (r^t, y^t).

6: Update $SSIterate$ by checking stopping criteria.

7: $\rho^{t+1}_{ij\omega} \leftarrow$ Adjust multipliers using (2.10).

8: $t \leftarrow t + 1$ ▷ Updating iteration counter.

9: **end while**

10: $(\hat{r}^m, \hat{y}^m) \leftarrow (r^{t-1}, y^{t-1}).$

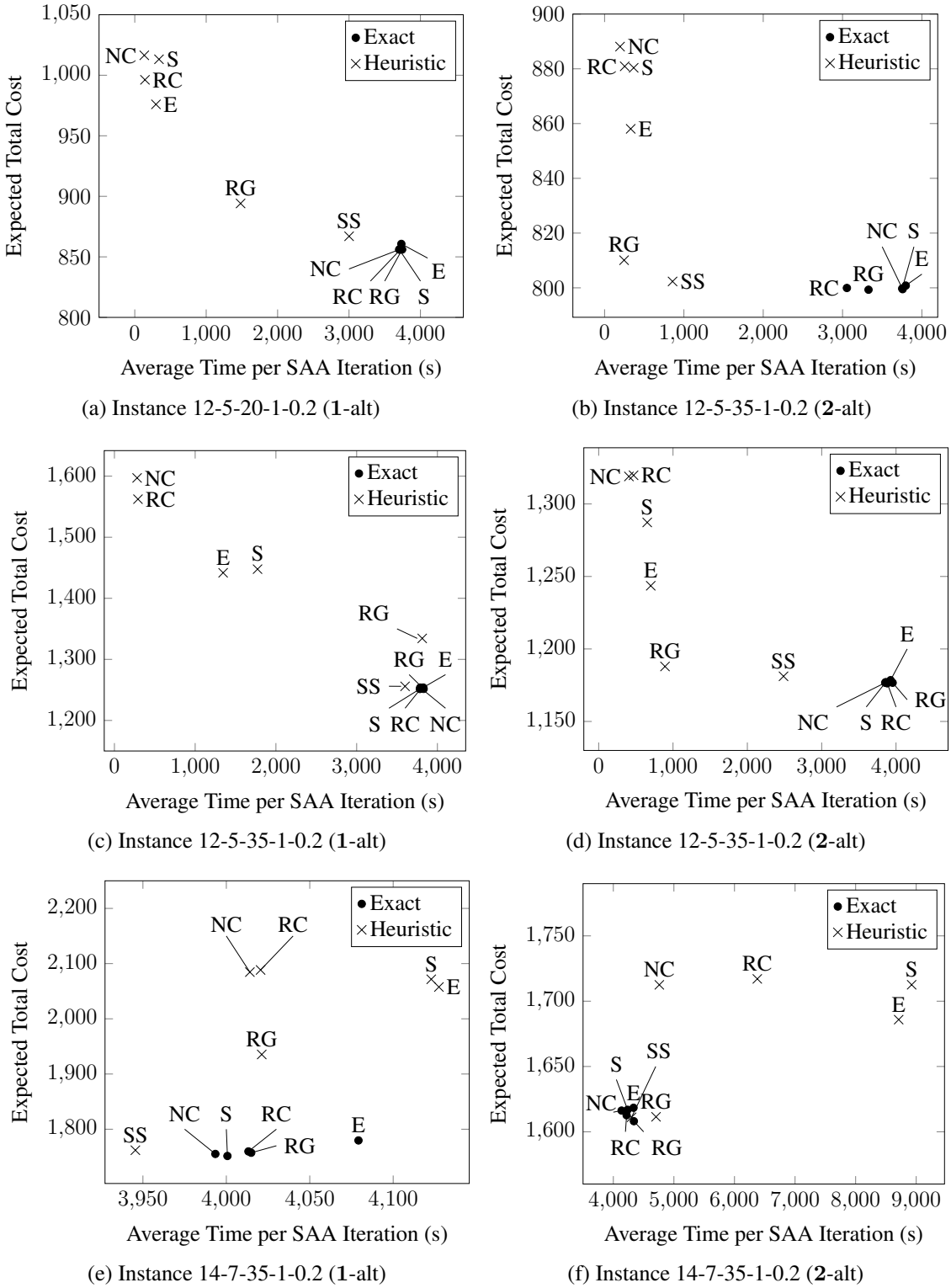
11: **return** $(\hat{r}^m, \hat{y}^m)$.

A.2 Comparison and Analysis of Solution Approaches

In this appendix, we present a more in-depth comparison of the exact and heuristics approaches discussed in Section 2.5. Specifically, we present a visual representations of the trade-off between solution quality and solution time, as well as the one between solution quality and the time to final design (TFD).

In Figure A.1, we plot for each instance and each method the average *total cost estimate* taken over the $M = 10$ SAA iterations against the average *time of an SAA iteration*. We define the *total cost estimate* of a sample problem as the first-stage cost of that sample problem plus the recourse cost estimate obtained by evaluating the first-stage design using $N' = 1000$ scenarios. The *time of an SAA iteration* is defined as the time to solve the sample problem plus the total evaluation time of all the $N' = 1000$ scenarios for that sample problem. Each row of plots in the figure corresponds to a single instance, with the

two plots in that row representing the different alt structures considered (**1**-alt and **2**-alt).



(a) Instance 12-5-20-1-0.2 (**1**-alt)

(b) Instance 12-5-35-1-0.2 (**2**-alt)

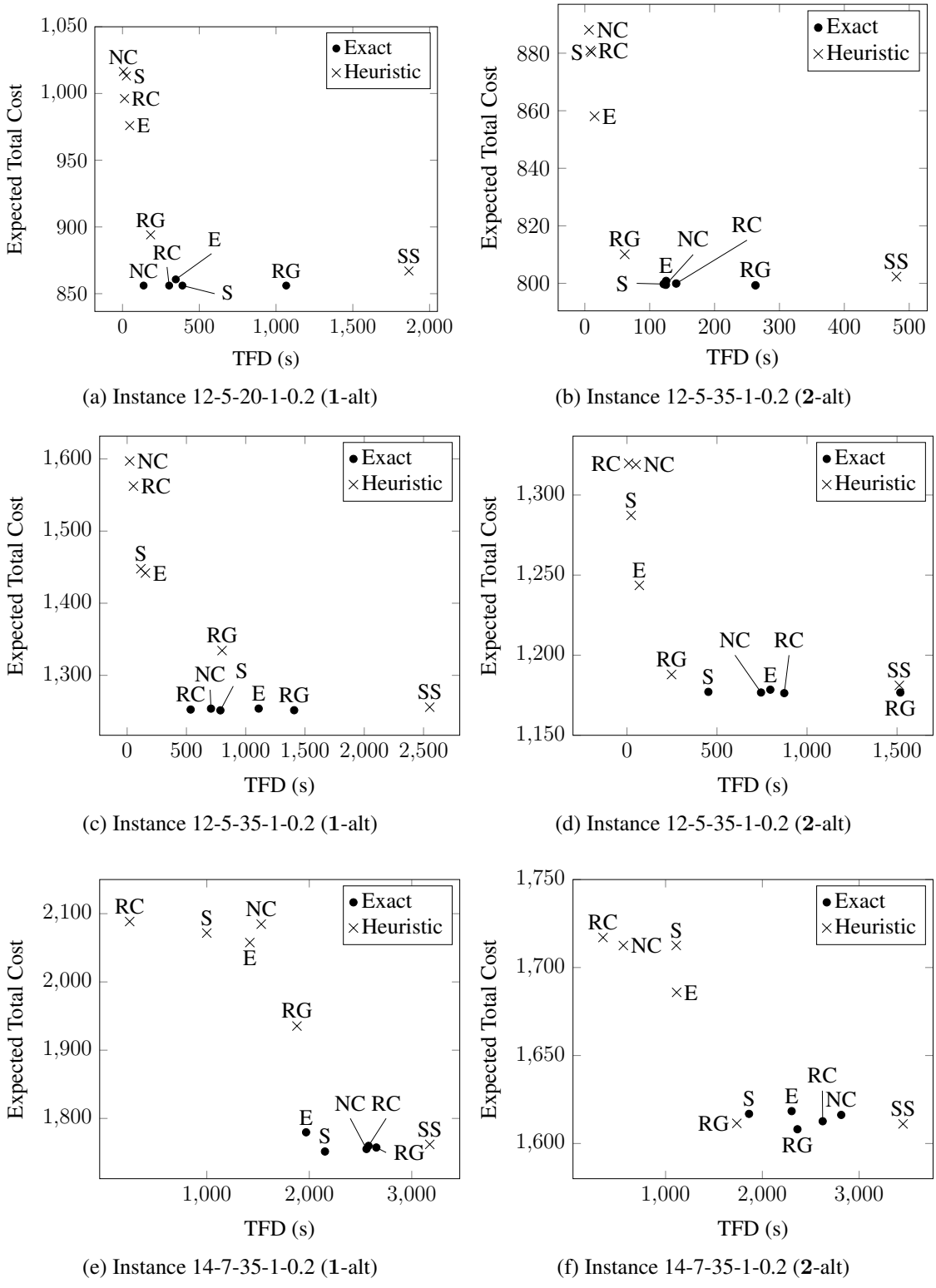
(c) Instance 12-5-35-1-0.2 (**1**-alt)

(d) Instance 12-5-35-1-0.2 (**2**-alt)

(e) Instance 14-7-35-1-0.2 (**1**-alt)

(f) Instance 14-7-35-1-0.2 (**2**-alt)

Figure A.1: Total expected cost vs. average time per SAA iteration



(a) Instance 12-5-20-1-0.2 (**1**-alt)

(b) Instance 12-5-35-1-0.2 (**2**-alt)

(c) Instance 12-5-35-1-0.2 (**1**-alt)

(d) Instance 12-5-35-1-0.2 (**2**-alt)

(e) Instance 14-7-35-1-0.2 (**1**-alt)

(f) Instance 14-7-35-1-0.2 (**2**-alt)

Figure A.2: Total expected cost vs. TFD

In terms of exact approaches, we point out that in Figure A.1 that there is very little variability in terms of these approaches in both solution time and solution quality. Due to the sizes of the instances considered and the fact that almost all the sample problems time out at the hour mark, this is not a surprising observation. In the case of Figure A.1b, for instance, the variability in time is a result of two of the approaches, namely Exact-RandCut and Exact-RGContraction, solving some of their sample problems to optimality before the hour mark, whereas the other three exhausted the time limit without being able to do so. This suggests that the two separation heuristics RandCut and RGContraction are effective separation heuristics for the cut inequalities, and have helped in the solution process.

Figure A.2 shows that there is slightly more variability in TFD among the five exact approaches. Generally, Exact-RGContraction is the slowest to settle on a final design, and this rarely comes with an increase in quality. In addition, Exact-Select seems to have a relatively low TFD while obtaining good quality designs (except perhaps in Figure A.2f).

For the heuristic approaches, the first thing to note from Figure A.1 is that there is much more variability in terms of solution quality and runtime among the different heuristic approaches compared to their exact counterparts. Moreover the average runtime of the heuristic approaches indicates that these heuristics can be faster than the exact approaches while still finding solutions of reasonable quality. In fact, on these instances, some heuristic approaches (e.g. Heuristic-SS) are comparable in solution quality to the exact approaches.

Not surprisingly, relaxing the integrality requirements for the second-stage variables without any method of compensating for that relaxation (NC) yields solutions that are usually much worse in quality compared to the exact approaches. In all six cases, Heuristic-NC is either the worst heuristic method in terms of solution quality or a very close second, and the difference in quality compared to the Heuristic-RG, Heuristic-E, Heuristic-S, and Heuristic-SS is significant. Furthermore, the algorithm Heuristic-RC doesn't seem to offer much improvement on the quality of the solution obtained by the Heuristic-NC approach. On the other hand, both of these algorithms are generally the among the fastest to run but

they lose their edge in terms of speed on the largest instance. This is partially explained by noting that in both settings of the smaller two instances, the sample problems were easy to solve to optimality, thereby allowing Heuristic-NC and Heuristic-RC to terminate before exhausting the time limit. In the 14-7-35-1-0.2 instance, the sample problems are generally a lot more difficult to solve in the hour time limit, and even Heuristic-NC and Heuristic-RC exhaust this limit (as is the case with the other four) causing all six methods to have more comparable solution times for the sample problems. The differences seen in Figures A.1e and A.1f are due to time taken by the evaluation subproblems. In Figure A.1f, we see a curious case in which Heuristic-E and Heuristic-S take a much longer time compared to the other four. Specifically, the evaluation subproblems for those two approaches took roughly 4.5 times the total time taken by the evaluation phase of Heuristic-NC. An interesting thing to note is that Heuristic-SS is the approach that consistently finds the best solution among heuristic methods. Furthermore, this quality does not necessarily require more computational time; SS is competitive in terms of average time in Figures A.1a–A.1d, but is actually the fastest heuristic method on the final two figures corresponding to the largest instance.

In Figure A.2, we again notice reasonable variability in TFD between the heuristic methods. The interesting comparison is between Heuristic-RG and Heuristic-SS. Despite Heuristic-SS consistently finding the best quality solution, it takes longer to reach this final design. On the other hand, Heuristic-RG achieves a much smaller TFD value sometimes sacrificing a little bit of solution quality in the process. The biggest drop-off in solution quality occurs in the $\mathbf{1}$-alt version of Instance 14-7-35-1-0.2 (Figure A.2e).

A.3 SAA Parameter Choices

For completeness, we include in Table A.1 the results of experimenting with the value of $M = 20$ (and a maximum of 30 minutes of computational time for each sample problem), compared with $M = 10$ (and a maximum of 60 minutes). The experiments were conducted using the instance 14-7-35-1-0.2, and we analyze both the $\mathbf{1}$-alt and the $\mathbf{2}$-alt models. In the

table, we present the total expected cost of the chosen design and the total computational time (in hours) taken by each.

Table A.1: Comparison of $M = 10$ and $M = 20$ sample problems in SAA

# of Replications	**1**-alt		**2**-alt	
	Expected Total Cost	Total Time (h)	Expected Total Cost	Total Time (h)
M=10	1729.77	11.30	1606.01	12.00
M=20	1740.88	12.61	1606.64	14.24

We observe that, in both models, despite the considerable increase in computational time, the design chosen in the case of $M = 10$ had a lower total expected cost. Based on these results, we chose $M = 10$ for the experiments in Section 2.6.4.

To justify our choice of $N = 10$ and to give an indication of the quality of solutions we obtained as a result of the SAA process, we include in Table A.2 the optimality gap estimates computed using Equation (2.5), and the percent optimality gap estimate (computed relative to the lower bound) for all 18 model-instance combinations.

Table A.2: Optimality gap estimates

Instance	**1**-alt		**2**-alt		Infinite-alt	
	Opt. Gap	% Gap	Opt. Gap	% Gap	Opt. Gap	% Gap
12-5-20-1-0.2	6.10	0.72%	2.77	0.35%	1.48	0.19%
12-5-20-1-0.6	10.71	1.23%	6.42	0.79%	6.44	0.79%
12-5-35-1-0.2	10.09	0.82%	4.16	0.36%	2.70	0.23%
12-5-35-1-0.6	24.09	1.90%	11.32	0.94%	8.61	0.72%
14-7-35-1-0.2	20.14	1.17%	-3.65	-0.23%	-4.64	-0.29%
14-7-35-1-0.6	22.30	1.24%	-0.13	-0.01%	1.17	0.07%

In this table, we note that all percent optimality gaps were less than 2% for all model-instance combinations. In fact, the majority of values were less than 1%, especially for the **2**-alt and Infinite-alt models. Using these observed values, we deemed that our solutions were of reasonable quality using the SAA parameter choices we selected.

REFERENCES

[1] T. G. Crainic, "Service network design in freight transportation," *European Journal of Operational Research*, vol. 122, no. 2, pp. 272–288, 2000.

[2] N. Wieberneit, "Service network design for freight transportation: A review," *OR Spectrum*, vol. 30, no. 1, pp. 77–112, 2008.

[3] B. Gendron, T. G. Crainic, and A. Frangioni, "Multicommodity capacitated network design," in *Telecommunications Network Planning*, B. Sansò and P. Soriano, Eds., 1999, pp. 1–19.

[4] A.-G. Lium, T. G. Crainic, and S. W. Wallace, "A study of demand stochasticity in service network design," *Transportation Science*, vol. 43, no. 2, pp. 144–157, May 2009.

[5] A. Baubaid, N. Boland, and M. Savelsbergh, "The value of limited flexibility in service network designs," *Transportation Science*, Oct. 2020.

[6] A. J. Kleywegt, A. Shapiro, and T. Homem-de-Mello, "The sample average approximation method for stochastic discrete optimization," *SIAM Journal on Optimization*, vol. 12, no. 2, pp. 479–502, 2002.

[7] I. Ghamlouche, T. G. Crainic, and M. Gendreau, "Cycle-based neighborhoods for fixed-charge capacitated multicommodity network design," *Operations Research*, vol. 51, no. 4, pp. 655–667, 2003.

[8] T. G. Crainic, M. Gendreau, and M. Farvolden Judith, "A simplex-based tabu search method for capacitated network design a simplex-based tabu search method for capacitated network design," *INFORMS Journal on Computing*, vol. 12, no. 3, pp. 223–236, 2000.

[9] R. Bai, S. W. Wallace, J. Li, and A. Y.-L. Chong, "Stochastic service network design with rerouting," *Transportation Research Part B: Methodological*, vol. 60, no. February, pp. 50–65, 2014.

[10] C.-C. Lin, "Load planning with uncertain demands for time-definite freight common carriers," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1873, no. 1, pp. 17–24, Jan. 2004.

[11] W. B. Powell and I. A. Koskosidis, "Shipment routing algorithms with tree constraints," *Transportation Science*, vol. 26, no. 3, pp. 230–245, Aug. 1992.

[12] G. Guastaroba, M. G. Speranza, and D. Vigo, "Intermediate facilities in freight transportation planning: A survey," *Transportation Science*, vol. 50, no. 3, pp. 763–789, 2016.

[13] M. Christiansen, K. Fagerholt, and D. Ronen, "Ship routing and scheduling: Status and perspectives," *Transportation Science*, vol. 38, no. 1, pp. 1–18, 2004.

[14] R. Agarwal and Ö. Ergun, "Ship scheduling and network design for cargo routing in liner shipping," *Transportation Science*, vol. 42, no. 2, pp. 175–196, 2008.

[15] L. B. Reinhardt and D. Pisinger, "A branch and cut algorithm for the container shipping network design problem," *Flexible Services and Manufacturing Journal*, vol. 24, no. 3, pp. 349–374, 2012.

[16] C. Barnhart and R. R. Schneur, "Air network design for express shipment service," *Operations Research*, vol. 44, no. 6, pp. 852–863, 1996.

[17] D. Kim, C. Barnhart, K. Ware, and G. Reinhardt, "Multimodal express package delivery: A service network design application," *Transportation Science*, vol. 33, no. 4, pp. 391–407, 1999.

[18] T. Grünert and H.-J. Sebastian, "Planning models for long-haul operations of postal and express shipment companies," *European Journal of Operational Research*, vol. 122, no. 2, pp. 289–309, 2000.

[19] C. Barnhart, N. Krishnan, D. Kim, and K. Ware, "Network design for express shipment delivery," *Computational Optimization and Applications*, vol. 21, no. 3, pp. 239–262, 2002.

[20] A. P. Armacost, C. Barnhart, and K. A. Ware, "Composite variable formulations for express shipment service network design," *Transportation Science*, vol. 36, no. 1, pp. 1–20, 2002.

[21] A. P. Armacost, C. Barnhart, K. A. Ware, and A. M. Wilson, "UPS optimizes its air network," *Interfaces*, vol. 34, no. 1, pp. 15–25, 2004.

[22] H. N. Newton, P. H. Vance, and C. Barnhart, "Constructing blocking plan to minimize handling costs," *Transportation Science*, vol. 32, no. 4, pp. 330–345, 1998.

[23] C. Barnhart, H. Jin, and P. H. Vance, "Railroad blocking: A network design application," *Operations Research*, vol. 48, no. 4, pp. 603–614, 2000.

[24] G Lulli, U Pietropaoli, and N Ricciardi, "Service network design for freight railway transportation: The italian case," *Journal of the Operational Research Society*, vol. 62, no. 12, pp. 2107–2119, 2011.

[25] E. Zhu, T. G. Crainic, and M. Gendreau, "Scheduled service network design for freight rail transportation," *Operations Research*, vol. 62, no. 2, pp. 383–400, 2014.

[26] T. G. Crainic and J.-M. Rousseau, "Multicommodity, multimode freight transportation: A general modeling and algorithmic framework for the service network design problem," *Transportation Research Part B*, vol. 20, no. 3, pp. 225–242, 1986.

[27] T. G. Crainic and J. Roy, "OR tools for tactical freight transportation planning," *European Journal of Operational Research*, vol. 33, no. 3, pp. 290–297, Feb. 1988.

[28] L. K. Nozick and E. K. Morlok, "A model for medium-term operations planning in an intermodal rail-truck service," *Transportation Research Part A: Policy and Practice*, vol. 31, no. 2, pp. 91–107, 1997.

[29] T. Yamada, B. F. Russ, J. Castro, and E. Taniguchi, "Designing multimodal freight transport networks: A heuristic approach and applications," *Transportation Science*, vol. 43, no. 2, pp. 129–143, 2009.

[30] D. Inghels, W. Dullaert, and D. Vigo, "A service network design model for multimodal municipal solid waste transport," *European Journal of Operational Research*, vol. 254, no. 1, pp. 68–79, 2016.

[31] T. L. Magnanti and R. T. Wong, "Network design and transportation planning models and algorithms," *Transportation Science*, vol. 18, no. 1, pp. 1–55, 1984.

[32] F. Barahona, "Network design using cut inequalities," *SIAM Journal on Optimization*, vol. 6, no. 3, pp. 823–837, 1996.

[33] D. Bienstock, S. Chopra, O. Günlük, and C.-Y. Tsai, "Minimum cost capacity installation for multicommodity network flows," *Mathematical Programming*, vol. 81, no. 2, pp. 177–199, 1998.

[34] O. Günlük, "A branch-and-cut algorithm for capacitated network design problems," *Mathematical Programming*, vol. 86, no. 1, pp. 17–39, 1999.

[35] A. Atamtürk, "On capacitated network design cut-set polyhedra," *Mathematical Programming, Series B*, vol. 92, no. 3, pp. 425–437, 2002.

[36] A. Frangioni and B. Gendron, "0-1 reformulations of the multicommodity capacitated network design problem," *Discrete Applied Mathematics*, vol. 157, no. 6, pp. 1229–1241, 2009.

[37] C. Raack, A. M. Koster, S. Orlowski, and R. Wessäly, "On cut-based inequalities for capacitated network design polyhedra," *Networks*, vol. 57, no. 2, pp. 141–156, 2011.

[38] M. Chouman, T. G. Crainic, and B. Gendron, "Commodity representations and cutset-based inequalities for multicommodity capacitated fixed-charge network design," *Transportation Science*, vol. 51, no. 2, pp. 650–667, 2017.

[39] I. Ghamlouche, T. G. Crainic, and M. Gendreau, "Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design," *Annals of Operations Research*, vol. 131, no. 1-4, pp. 109–133, 2004.

[40] A. Erera, M. Hewitt, M. Savelsbergh, and Y. Zhang, "Improved load plan design through integer programming based local search," *Transportation Science*, vol. 47, no. 3, pp. 412–427, Aug. 2013.

[41] K. Lindsey, A. Erera, and M. Savelsbergh, "Improved integer programming-based neighborhood search for less-than-truckload load plan design," *Transportation Science*, vol. 50, no. 4, pp. 1360–1379, 2016.

[42] D. Kim and P. M. Pardalos, "A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure," *Operations Research Letters*, vol. 24, no. 4, pp. 195–203, 1999.

[43] T. G. Crainic, B. Gendron, and G. Hernu, "A slope scaling/Lagrangean perturbation heuristic with long-term memory for multicommodity capacitated fixed-charge network design," *Journal of Heuristics*, vol. 10, no. 5, pp. 525–545, Sep. 2004.

[44] A. I. Jarrah, E. Johnson, and L. C. Neubert, "Large-scale, less-than-truckload service network design," *Operations Research*, vol. 57, no. 3, pp. 609–625, 2009.

[45] T. G. Crainic, X. Fu, M. Gendreau, W. Rei, and S. W. Wallace, "Progressive hedging-based meta-heuristics for stochastic network design," *Networks*, vol. 58, no. 2, pp. 114–124, 2011.

[46] A. Hoff, A. G. Lium, A. Løkketangen, and T. G. Crainic, "A metaheuristic for stochastic service network design," *Journal of Heuristics*, vol. 16, no. 5, pp. 653–679, 2010.

[47] X. Wang and S. W. Wallace, "Stochastic scheduled service network design in the presence of a spot market for excess capacity," *EURO Journal on Transportation and Logistics*, vol. 5, no. 4, pp. 393–413, 2016.

[48] C. Sun, S. W. Wallace, and L. Luo, "Stochastic multi-commodity network design: The quality of deterministic solutions," *Operations Research Letters*, vol. 45, no. 3, pp. 266–268, 2017.

[49] X. Wang, T. G. Crainic, and S. W. Wallace, "Stochastic network design for planning scheduled transportation services: The value of deterministic solutions," *INFORMS Journal on Computing*, vol. 31, no. 1, pp. 153–170, 2019.

[50] W. B. Powell and Y. Sheffi, "The load planning problem of motor carriers: Problem description and a proposed solution approach," *Transportation Research Part A: General*, vol. 17, no. 6, pp. 471–480, 1983.

[51] W. B. Powell, "A local improvement heuristic for the design of less-than-truckload motor carrier networks," *Transportation Science*, vol. 20, no. 4, pp. 246–257, Nov. 1986.

[52] W. B. Powell and Y. Sheffi, "OR practice—design and implementation of an interactive optimization system for network design in the motor carrier industry," *Operations Research*, vol. 37, no. 1, pp. 12–29, Feb. 1989.

[53] C. C. Lin, "The freight routing problem of time-definite freight delivery common carriers," *Transportation Research Part B: Methodological*, vol. 35, no. 6, pp. 525–547, 2001.

[54] C. C. Lin and J. S. J. Lin, "The multistage stochastic integer load planning problem," *Transportation Research Part E: Logistics and Transportation Review*, vol. 43, no. 2, pp. 143–156, 2007.

[55] W. C. Jordan and S. C. Graves, "Principles on the benefits of manufacturing process flexibility," *Management science*, vol. 41, no. 4, pp. 577–594, 1995.

[56] M. W. Ulmer and B. W. Thomas, "Meso-parametric value function approximation for dynamic customer acceptances in delivery routing," *European Journal of Operational Research*, vol. 285, no. 1, pp. 183–195, Apr. 2019.

[57] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, "A review of dynamic vehicle routing problems," *European Journal of Operational Research*, vol. 225, no. 1, pp. 1–11, Feb. 2013.

[58] U. Ritzinger, J. Puchinger, and R. F. Hartl, "A survey on dynamic and stochastic vehicle routing problems," *International Journal of Production Research*, vol. 54, no. 1, pp. 215–231, Jan. 2016.

[59] C. B. Cunha and M. R. Silva, "A genetic algorithm for the problem of configuring a hub-and-spoke network for a LTL trucking company in brazil," *European Journal of Operational Research*, vol. 179, no. 3, pp. 747–758, Jun. 2007.

[60] J. F. Campbell, "Hub location for time definite transportation," *Computers & Operations Research*, New Developments on Hub Location, vol. 36, no. 12, pp. 3107–3116, Dec. 2009.

[61] C.-C. Lin and S.-C. Lee, "Hub network design problem with profit optimization for time-definite LTL freight transportation," *Transportation Research Part E: Logistics and Transportation Review*, vol. 114, pp. 104–120, Jun. 2018.

[62] S. Alumur and B. Y. Kara, "Network hub location problems: The state of the art," *European Journal of Operational Research*, vol. 190, no. 1, pp. 1–21, Oct. 2008.

[63] J. F. Campbell and M. E. O'Kelly, "Twenty-five years of hub location research," *Transportation Science*, vol. 46, no. 2, pp. 153–169, May 2012.

[64] W. B. Powell, P. Jaillet, and A. Odoni, "Stochastic and dynamic networks and routing," in *Handbooks in Operations Research and Management Science*, ser. Network Routing, vol. 8, Elsevier, Jan. 1995, pp. 141–295.

[65] W. B. Powell, "Dynamic models of transportation operations," in *Handbooks in Operations Research and Management Science*, vol. 11, Elsevier, 2003, pp. 677–756, ISBN: 978-0-444-51328-1.

[66] A. J. Kleywegt and J. D. Papastavrou, "Acceptance and dispatching policies for a distribution problem," *Transportation Science*, vol. 32, no. 2, pp. 127–141, May 1998.

[67] R. Cheung and B. Muralidharan, "Impact of dynamic decision making on hub-and-spoke freight transportation networks," *Annals of Operations Research*, vol. 87, pp. 49–71, Apr. 1999.

[68] R. K. Cheung and B. Muralidharan, "Dynamic routing for priority shipments in LTL service networks," *Transportation Science*, vol. 34, no. 1, pp. 86–98, Feb. 2000.

[69] N Shi, R. K. Cheung, H Xu, and K. K. Lai, "An adaptive routing strategy for freight transportation networks," *Journal of the Operational Research Society*, vol. 62, no. 4, pp. 799–805, Apr. 2011.

[70] A. Erera, B. Karacık, and M. Savelsbergh, "A dynamic driver management scheme for less-than-truckload carriers," *Computers & Operations Research*, Part Special Issue: Topics in Real-Time Supply Chain Management, vol. 35, no. 11, pp. 3397–3411, Nov. 2008.

[71] A. L. Erera, M. Hewitt, M. W. P. Savelsbergh, and Y. Zhang, "Creating schedules and computing operating costs for LTL load plans," *Computers & Operations Research*, Transport Scheduling, vol. 40, no. 3, pp. 691–702, Mar. 2013.

[72] B. Hejazi and A. Haghani, "Dynamic decision making for less-than-truckload trucking operations," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2032, no. 1, pp. 17–25, Jan. 2007.

[73] H. P. Simao and W. B. Powell, "Decomposition methods for dynamic load planning and driver management in LTL trucking," in *7th INFORMS Transportation Science and Logistics Society Workshop*, Vienna, 2019.

[74] Y. Ridouane, I. Herszterg, N. Boland, and A. Erera, "Near real-time loadplan adjustments for less-than-truckload carriers," *Optimization Online*, 2020.

[75] W. B. Powell, H. P. Simao, and B. Bouzaiene-Ayari, "Approximate dynamic programming in transportation and logistics: A unified framework," *EURO Journal on Transportation and Logistics*, vol. 1, no. 3, pp. 237–284, Sep. 2012.

[76] W. B. Powell, J. A. Shapiro, and H. P. Simão, "An adaptive dynamic programming algorithm for the heterogeneous resource allocation problem," *Transportation Science*, vol. 36, no. 2, pp. 231–249, May 2002.

[77] M. W. Ulmer, D. C. Mattfeld, and F. Köster, "Budgeting time for dynamic vehicle routing with stochastic customer requests," *Transportation Science*, vol. 52, no. 1, pp. 20–37, Mar. 2017.

[78] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, 2nd ed, ser. Wiley Series in Probability and Statistics. Hoboken, N.J: Wiley, 2011, ISBN: 978-0-470-60445-8.

[79] H. Topaloglu and W. B. Powell, "Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems," *INFORMS Journal on Computing*, vol. 18, no. 1, pp. 31–42, Feb. 2006.

[80] L. C. Dall'Orto, T. G. Crainic, J. E. Leal, and W. B. Powell, "The single-node dynamic service scheduling and dispatching problem," *European Journal of Operational Research*, vol. 170, no. 1, pp. 1–23, Apr. 2006.

[81] W. van Heeswijk, M. Mes, and M. Schutten, "An approximate dynamic programming approach to urban freight distribution with batch arrivals," in *Computational Logistics*, F. Corman, S. Voß, and R. R. Negenborn, Eds., ser. Lecture Notes in Computer Science, vol. 9335, Cham: Springer International Publishing, 2015, pp. 61–75, ISBN: 978-3-319-24264-4.

[82] W. J. A. van Heeswijk, M. R. K. Mes, and J. M. J. Schutten, "The delivery dispatching problem with time windows for urban consolidation centers," *Transportation Science*, vol. 53, no. 1, pp. 203–221, Feb. 2019.

[83] A. E. Pérez Rivera and M. R. K. Mes, "Anticipatory freight selection in intermodal long-haul round-trips," *Transportation Research Part E: Logistics and Transportation Review*, vol. 105, pp. 176–194, Sep. 2017.

[84] A. Toriello, G. Nemhauser, and M. Savelsbergh, "Decomposing inventory routing problems with approximate value functions," *Naval Research Logistics (NRL)*, vol. 57, no. 8, pp. 718–727, 2010.

[85] M. W. Ulmer, N. Soeffker, and D. C. Mattfeld, "Value function approximation for dynamic multi-period vehicle routing," *European Journal of Operational Research*, vol. 269, no. 3, pp. 883–899, Sep. 2018.

[86] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second edition, ser. Adaptive Computation and Machine Learning Series. Cambridge, Massachusetts: The MIT Press, 2018, ISBN: 978-0-262-03924-6.

[87] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, ser. Optimization and Neural Computation Series. Belmont, Mass: Athena Scientific, 1996, ISBN: 978-1-886529-10-6.

[88] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015.

[89] W. van Heeswijk and H. La Poutré, "Approximate dynamic programming with neural networks in linear discrete action spaces," Feb. 2019. arXiv: 1902.09855 [cs.LG].

[90] F. He, J. Yang, and M. Li, "Vehicle scheduling under stochastic trip times: An approximate dynamic programming approach," *Transportation Research Part C: Emerging Technologies*, vol. 96, pp. 144–159, Nov. 2018.

[91] W. Van Heeswijk and H. La Poutré, "Scalability and performance of decentralized planning in flexible transport networks," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct. 2018, pp. 292–297.

[92] A. Delarue, R. Anderson, and C. Tjandraatmadja, "Reinforcement learning with combinatorial actions: An application to vehicle routing," Oct. 2020. arXiv: 2010.12001 [cs.LG].

[93] D. Bertsimas and R. Demir, "An approximate dynamic programming approach to multidimensional knapsack problems," *Management Science*, vol. 48, no. 4, pp. 550–565, Apr. 2002.

[94] J. O. Riis, "Discounted markov programming in a periodic process," *Operations Research*, vol. 13, no. 6, pp. 920–929, Dec. 1965.

[95] L. M. M. Veugen, J. van der Wal, and J. Wessels, "The numerical exploitation of periodicity in markov decision processes," *Operations-Research-Spektrum*, vol. 5, no. 2, pp. 97–103, Jun. 1983.

[96] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, ser. Wiley Series in Probability and Statistics. Hoboken, NJ: Wiley-Interscience, 2005, ISBN: 978-0-471-72782-8.

[97] C. C. Kokonendji and S. S. Zocchi, "Extensions of discrete triangular distributions and boundary bias in kernel estimation for discrete functions," *Statistics & Probability Letters*, vol. 80, no. 21, pp. 1655–1662, Nov. 2010.

[98] C. M. Bishop, *Pattern Recognition and Machine Learning*, ser. Information Science and Statistics. New York: Springer, 2006, ISBN: 978-0-387-31073-2.

[99] S. S. Haykin and S. S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. New York: Prentice Hall, 2009, ISBN: 978-0-13-147139-9.

[100] R. R. Bunel, I. Turkaslan, P. Torr, P. Kohli, and P. K. Mudigonda, "A unified view of piecewise linear neural network verification," in *Advances in Neural Information Processing Systems*, 2018, pp. 4790–4799.

[101] M. Fischetti and J. Jo, "Deep neural networks and mixed integer linear optimization," *Constraints*, vol. 23, no. 3, pp. 296–309, Jul. 2018.

[102] R. Anderson, J. Huchette, W. Ma, C. Tjandraatmadja, and J. P. Vielma, "Strong mixed-integer programming formulations for trained neural networks," *Mathematical Programming*, vol. 183, pp. 3–39, Feb. 2020.

[103] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," in *Neural Networks: Tricks of the Trade: Second Edition*, ser. Lecture Notes in Computer Science, G. Montavon, G. B. Orr, and K.-R. Müller, Eds., Berlin, Heidelberg: Springer, 2012, pp. 9–48, ISBN: 978-3-642-35289-8.

[104] G. James, D. Witten, T. Hastie, and R. Tibshirani, Eds., *An Introduction to Statistical Learning: With Applications in R*, ser. Springer Texts in Statistics 103. New York: Springer, 2013, ISBN: 978-1-4614-7137-0.

[105] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.